Arm Performance Libraries Reference Guide

Version 19.2.0

Document Number 101004_1920_00

Release Information

Issue	Date	Confidentiality	Change
1830-00	20 June 2018	Non-Confidential	Document release for Arm Performance Libraries version 18.3
1840-00	17 July 2018	Non-Confidential	Update for Arm Performance Libraries version 18.4
1900-00	2 November 2018	Non-Confidential	Update for Arm Performance Libraries version 19.0
1910-00	8 March 2019	Non-Confidential	Update for Arm Performance Libraries version 19.1
1920-00	07 June 2019	Non-Confidential	Update for Arm Performance Libraries version 19.2

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349

Confidentiality Status

Product Status

The information in this document is Final, that is for a developed product.

Web Address

http://www.arm.com

Contents

1 Introduction **3**
1.1 Arm Performance Libraries . . . 3
1.2 References . . . 3

2 General Information **5**
2.1 Accessing the Library . . . 5
2.2 Arm Performance Libraries Fortran and C interfaces . . . 6
2.3 Arm Performance Libraries variant using 64-bit integer (INTEGER*8) arguments . . . 7
2.4 Library Version and Build Information . . . 7
2.4.1 Fortran specifications . . . 7
2.4.2 C specifications . . . 7
2.5 Example programs calling Arm Performance Libraries . . . 7

3 BLAS Basic Linear Algebra Subprograms **9**
3.1 Overview . . . 9
3.2 BLAS level 1 . . . 10
3.2.1 caxpy . . . 10
3.2.2 ccopy . . . 11
3.2.3 cdotc . . . 12
3.2.4 cdotu . . . 13
3.2.5 crot . . . 14
3.2.6 crotg . . . 15
3.2.7 cscal . . . 16
3.2.8 csrot . . . 17
3.2.9 csscal . . . 18
3.2.10 cswap . . . 19
3.2.11 dasum . . . 20
3.2.12 daxpy . . . 20
3.2.13 dcopy . . . 21
3.2.14 ddot . . . 22
3.2.15 dnrm2 . . . 23
3.2.16 drot . . . 24
3.2.17 drotg . . . 25
3.2.18 drotm . . . 26
3.2.19 drotmg . . . 27
3.2.20 dscal . . . 28
3.2.21 dsdot . . . 29
3.2.22 dswap . . . 30
3.2.23 dzasum . . . 30
3.2.24 dznrm2 . . . 31
3.2.25 icamax . . . 32
3.2.26 idamax . . . 33
3.2.27 isamax . . . 33
3.2.28 izamax . . . 34
3.2.29 sasum . . . 35
3.2.30 saxpy . . . 36
3.2.31 scasum . . . 37
3.2.32 scnrm2 . . . 37

3.2.33 scopy . . . 38
3.2.34 sdot . . . 39
3.2.35 sdsdot . . . 40
3.2.36 snrm2 . . . 41
3.2.37 srot . . . 42
3.2.38 srotg . . . 43
3.2.39 srotm . . . 44
3.2.40 srotmg . . . 45
3.2.41 sscal . . . 46
3.2.42 sswap . . . 47
3.2.43 zaxpy . . . 47
3.2.44 zcopy . . . 48
3.2.45 zdotc . . . 49
3.2.46 zdotu . . . 50
3.2.47 zdrot . . . 51
3.2.48 zdscal . . . 53
3.2.49 zrot . . . 53
3.2.50 zrotg . . . 55
3.2.51 zscal . . . 55
3.2.52 zswap . . . 56
3.3 BLAS level 2 . . . 57
3.3.1 cgbmv . . . 57
3.3.2 cgemv . . . 59
3.3.3 cgerc . . . 61
3.3.4 cgeru . . . 62
3.3.5 chbmv . . . 64
3.3.6 chemv . . . 66
3.3.7 cher . . . 68
3.3.8 cher2 . . . 69
3.3.9 chpmv . . . 70
3.3.10 chpr . . . 72
3.3.11 chpr2 . . . 73
3.3.12 cspmv . . . 75
3.3.13 cspr . . . 76
3.3.14 csymv . . . 78
3.3.15 csyr . . . 79
3.3.16 ctbmv . . . 81
3.3.17 ctbsv . . . 83
3.3.18 ctpmv . . . 85
3.3.19 ctpsv . . . 86
3.3.20 ctrmv . . . 88
3.3.21 ctrsv . . . 89
3.3.22 dgbmv . . . 91
3.3.23 dgemv . . . 93
3.3.24 dger . . . 94
3.3.25 dsbmv . . . 96
3.3.26 dspmv . . . 98
3.3.27 dspr . . . 99
3.3.28 dspr2 . . . 100
3.3.29 dsymv . . . 102
3.3.30 dsyr . . . 103
3.3.31 dsyr2 . . . 105
3.3.32 dtbmv . . . 106
3.3.33 dtbsv . . . 108
3.3.34 dtpmv . . . 110
3.3.35 dtpsv . . . 112
3.3.36 dtrmv . . . 113
3.3.37 dtrsv . . . 115

3.3.38 sgbmv . . . 116
3.3.39 sgemv . . . 118
3.3.40 sger . . . 120
3.3.41 ssbmv . . . 121
3.3.42 sspmv . . . 123
3.3.43 sspr . . . 125
3.3.44 sspr2 . . . 126
3.3.45 ssymv . . . 128
3.3.46 ssyr . . . 129
3.3.47 ssyr2 . . . 130
3.3.48 stbmv . . . 132
3.3.49 stbsv . . . 134
3.3.50 stpmv . . . 136
3.3.51 stpsv . . . 137
3.3.52 strmv . . . 139
3.3.53 strsv . . . 140
3.3.54 zgbmv . . . 142
3.3.55 zgemv . . . 144
3.3.56 zgerc . . . 146
3.3.57 zgeru . . . 147
3.3.58 zhbmv . . . 148
3.3.59 zhemv . . . 151
3.3.60 zher . . . 152
3.3.61 zher2 . . . 154
3.3.62 zhpmv . . . 155
3.3.63 zhpr . . . 157
3.3.64 zhpr2 . . . 158
3.3.65 zspmv . . . 159
3.3.66 zspr . . . 161
3.3.67 zsymv . . . 162
3.3.68 zsyr . . . 164
3.3.69 ztbmv . . . 165
3.3.70 ztbsv . . . 167
3.3.71 ztpmv . . . 169
3.3.72 ztpsv . . . 171
3.3.73 ztrmv . . . 172
3.3.74 ztrsv . . . 174
3.4 BLAS level 3 . . . 175
3.4.1 cgemm . . . 175
3.4.2 chemm . . . 178
3.4.3 cher2k . . . 180
3.4.4 cherk . . . 182
3.4.5 csymm . . . 183
3.4.6 csyr2k . . . 185
3.4.7 csyrk . . . 187
3.4.8 ctrmm . . . 189
3.4.9 ctrsm . . . 191
3.4.10 dgemm . . . 193
3.4.11 dsymm . . . 195
3.4.12 dsyr2k . . . 197
3.4.13 dsyrk . . . 199
3.4.14 dtrmm . . . 201
3.4.15 dtrsm . . . 203
3.4.16 sgemm . . . 205
3.4.17 ssymm . . . 207
3.4.18 ssyr2k . . . 209
3.4.19 ssyrk . . . 211
3.4.20 strmm . . . 213

3.4.21 strsm . . . 215
3.4.22 zgemm . . . 217
3.4.23 zhemm . . . 219
3.4.24 zher2k . . . 221
3.4.25 zherk . . . 223
3.4.26 zsymm . . . 225
3.4.27 zsyr2k . . . 227
3.4.28 zsyrk . . . 229
3.4.29 ztrmm . . . 230
3.4.30 ztrsm . . . 232
3.5 CBLAS functions . . . 234
3.5.1 cblas_caxpby . . . 234
3.5.2 cblas_caxpy . . . 235
3.5.3 cblas_ccopy . . . 236
3.5.4 cblas_cgbmv . . . 236
3.5.5 cblas_cgemm . . . 237
3.5.6 cblas_cgemm3m . . . 237
3.5.7 cblas_cgemv . . . 238
3.5.8 cblas_cgerc . . . 239
3.5.9 cblas_cgeru . . . 239
3.5.10 cblas_chbmv . . . 240
3.5.11 cblas_chemm . . . 240
3.5.12 cblas_chemv . . . 241
3.5.13 cblas_cher . . . 242
3.5.14 cblas_cher2 . . . 242
3.5.15 cblas_cher2k . . . 243
3.5.16 cblas_cherk . . . 243
3.5.17 cblas_chpmv . . . 244
3.5.18 cblas_chpr . . . 244
3.5.19 cblas_chpr2 . . . 245
3.5.20 cblas_cscal . . . 245
3.5.21 cblas_csscal . . . 246
3.5.22 cblas_cswap . . . 247
3.5.23 cblas_csymm . . . 247
3.5.24 cblas_csyr2k . . . 248
3.5.25 cblas_csyrk . . . 248
3.5.26 cblas_ctbmv . . . 249
3.5.27 cblas_ctbsv . . . 250
3.5.28 cblas_ctpmv . . . 250
3.5.29 cblas_ctpsv . . . 251
3.5.30 cblas_ctrmm . . . 251
3.5.31 cblas_ctrmv . . . 252
3.5.32 cblas_ctrsm . . . 253
3.5.33 cblas_ctrsv . . . 253
3.5.34 cblas_cwaxpby . . . 254
3.5.35 cblas_dasum . . . 254
3.5.36 cblas_caxpby . . . 255
3.5.37 cblas_daxpy . . . 255
3.5.38 cblas_dcopy . . . 256
3.5.39 cblas_ddot . . . 257
3.5.40 cblas_dgbmv . . . 257
3.5.41 cblas_dgemm . . . 258
3.5.42 cblas_dgemv . . . 258
3.5.43 cblas_dger . . . 259
3.5.44 cblas_dnrm2 . . . 260
3.5.45 cblas_drot . . . 260
3.5.46 cblas_drotg . . . 261
3.5.47 cblas_drotm . . . 261

3.5.48 cblas_drotmg . . . 262
3.5.49 cblas_dsbmv . . . 262
3.5.50 cblas_dscal . . . 263
3.5.51 cblas_dsdot . . . 263
3.5.52 cblas_dspmv . . . 264
3.5.53 cblas_dspr . . . 265
3.5.54 cblas_dspr2 . . . 265
3.5.55 cblas_dswap . . . 266
3.5.56 cblas_dsymm . . . 266
3.5.57 cblas_dsymv . . . 267
3.5.58 cblas_dsyr . . . 267
3.5.59 cblas_dsyr2 . . . 268
3.5.60 cblas_dsyr2k . . . 269
3.5.61 cblas_dsyrk . . . 269
3.5.62 cblas_dtbmv . . . 270
3.5.63 cblas_dtbsv . . . 270
3.5.64 cblas_dtpmv . . . 271
3.5.65 cblas_dtpsv . . . 272
3.5.66 cblas_dtrmm . . . 272
3.5.67 cblas_dtrmv . . . 273
3.5.68 cblas_dtrsm . . . 273
3.5.69 cblas_dtrsv . . . 274
3.5.70 cblas_dwaxpby . . . 274
3.5.71 cblas_dzasum . . . 275
3.5.72 cblas_dznrm2 . . . 276
3.5.73 cblas_icamax . . . 276
3.5.74 cblas_idamax . . . 277
3.5.75 cblas_isamax . . . 277
3.5.76 cblas_izamax . . . 278
3.5.77 cblas_sasum . . . 278
3.5.78 cblas_saxpby . . . 279
3.5.79 cblas_saxpy . . . 279
3.5.80 cblas_scasum . . . 280
3.5.81 cblas_scnrm2 . . . 280
3.5.82 cblas_scopy . . . 281
3.5.83 cblas_sdot . . . 281
3.5.84 cblas_sdsdot . . . 282
3.5.85 cblas_sgbmv . . . 282
3.5.86 cblas_sgemm . . . 283
3.5.87 cblas_sgemv . . . 284
3.5.88 cblas_sger . . . 284
3.5.89 cblas_snrm2 . . . 285
3.5.90 cblas_srot . . . 285
3.5.91 cblas_srotg . . . 286
3.5.92 cblas_srotm . . . 286
3.5.93 cblas_srotmg . . . 287
3.5.94 cblas_ssbmv . . . 287
3.5.95 cblas_sscal . . . 288
3.5.96 cblas_sspmv . . . 289
3.5.97 cblas_sspr . . . 289
3.5.98 cblas_sspr2 . . . 290
3.5.99 cblas_sswap . . . 290
3.5.100 cblas_ssymm . . . 291
3.5.101 cblas_ssymv . . . 291
3.5.102 cblas_ssyr . . . 292
3.5.103 cblas_ssyr2 . . . 292
3.5.104 cblas_ssyr2k . . . 293
3.5.105 cblas_ssyrk . . . 294

3.5.106 cblas_stbmv . . . 294
3.5.107 cblas_stbsv . . . 295
3.5.108 cblas_stpmv . . . 295
3.5.109 cblas_stpsv . . . 296
3.5.110 cblas_strmm . . . 297
3.5.111 cblas_strmv . . . 297
3.5.112 cblas_strsm . . . 298
3.5.113 cblas_strsv . . . 298
3.5.114 cblas_swaxpby . . . 299
3.5.115 cblas_xerbla . . . 300
3.5.116 cblas_zaxpby . . . 300
3.5.117 cblas_zaxpy . . . 301
3.5.118 cblas_zcopy . . . 301
3.5.119 cblas_zdscal . . . 302
3.5.120 cblas_zgbmv . . . 302
3.5.121 cblas_zgemm . . . 303
3.5.122 cblas_zgemm3m . . . 304
3.5.123 cblas_zgemv . . . 304
3.5.124 cblas_zgerc . . . 305
3.5.125 cblas_zgeru . . . 305
3.5.126 cblas_zhbmv . . . 306
3.5.127 cblas_zhemm . . . 307
3.5.128 cblas_zhemv . . . 307
3.5.129 cblas_zher . . . 308
3.5.130 cblas_zher2 . . . 308
3.5.131 cblas_zher2k . . . 309
3.5.132 cblas_zherk . . . 310
3.5.133 cblas_zhpmv . . . 310
3.5.134 cblas_zhpr . . . 311
3.5.135 cblas_zhpr2 . . . 311
3.5.136 cblas_zscal . . . 312
3.5.137 cblas_zswap . . . 312
3.5.138 cblas_zsymm . . . 313
3.5.139 cblas_zsyr2k . . . 314
3.5.140 cblas_zsyrk . . . 314
3.5.141 cblas_ztbmv . . . 315
3.5.142 cblas_ztbsv . . . 316
3.5.143 cblas_ztpmv . . . 316
3.5.144 cblas_ztpsv . . . 317
3.5.145 cblas_ztrmm . . . 317
3.5.146 cblas_ztrmv . . . 318
3.5.147 cblas_ztrsm . . . 319
3.5.148 cblas_ztrsv . . . 319
3.5.149 cblas_zwaxpby . . . 320
3.6 BLAS Extensions . . . 320
3.6.1 caxpby . . . 320
3.6.2 cgemm3m . . . 322
3.6.3 cwaxpby . . . 324
3.6.4 daxpby . . . 325
3.6.5 dwaxpby . . . 326
3.6.6 hgemm . . . 328
3.6.7 saxpby . . . 330
3.6.8 swaxpby . . . 331
3.6.9 zaxpby . . . 332
3.6.10 zgemm3m . . . 333
3.6.11 zwaxpby . . . 335
3.6.12 sgemm_batch . . . 337
3.6.13 dgemm_batch . . . 340

3.6.14 cgemm_batch . . . 342
3.6.15 zgemm_batch . . . 345

4 LAPACK Linear Algebra Package 349
4.1 Introduction to LAPACK . . . 349
4.2 Reference sources for LAPACK . . . 349
4.3 LAPACK matrix factorization routines . . . 350
4.3.1 cgbtrf . . . 350
4.3.2 cgelq . . . 351
4.3.3 cgelqf . . . 353
4.3.4 cgelqt . . . 354
4.3.5 cgemlq . . . 356
4.3.6 cgemlqt . . . 357
4.3.7 cgemqr . . . 359
4.3.8 cgemqrt . . . 361
4.3.9 cgeqlf . . . 363
4.3.10 cgeqp3 . . . 364
4.3.11 cgeqr . . . 366
4.3.12 cgeqrf . . . 367
4.3.13 cgeqrfp . . . 369
4.3.14 cgeqrt . . . 370
4.3.15 cgerqf . . . 371
4.3.16 cgetrf . . . 373
4.3.17 cgetrf2 . . . 374
4.3.18 cggqrf . . . 375
4.3.19 cggrqf . . . 378
4.3.20 cgttrf . . . 380
4.3.21 chetrf . . . 381
4.3.22 chetrf_aa . . . 383
4.3.23 chetrf_rk . . . 384
4.3.24 chetrf_rook . . . 387
4.3.25 chptrf . . . 388
4.3.26 cpbtrf . . . 389
4.3.27 cpftrf . . . 391
4.3.28 cpotrf . . . 392
4.3.29 cpotrf2 . . . 394
4.3.30 cpptrf . . . 395
4.3.31 cpstrf . . . 396
4.3.32 cpttrf . . . 397
4.3.33 csptrf . . . 398
4.3.34 csytrf . . . 400
4.3.35 csytrf_aa . . . 401
4.3.36 csytrf_rk . . . 403
4.3.37 csytrf_rook . . . 405
4.3.38 ctplqt . . . 407
4.3.39 ctpmlqt . . . 408
4.3.40 ctpmqrt . . . 410
4.3.41 ctpqrt . . . 413
4.3.42 ctzrzf . . . 414
4.3.43 cunglq . . . 416
4.3.44 cungql . . . 417
4.3.45 cungqr . . . 419
4.3.46 cungrq . . . 420
4.3.47 cunmlq . . . 421
4.3.48 cunmql . . . 423
4.3.49 cunmqr . . . 425
4.3.50 cunmrq . . . 427
4.3.51 cunmrz . . . 429

4.3.52 dgbtrf . . . 431
4.3.53 dgelq . . . 433
4.3.54 dgelqf . . . 434
4.3.55 dgelqt . . . 435
4.3.56 dgemlq . . . 437
4.3.57 dgemlqt . . . 439
4.3.58 dgemqr . . . 441
4.3.59 dgemqrt . . . 442
4.3.60 dgeqlf . . . 444
4.3.61 dgeqp3 . . . 446
4.3.62 dgeqr . . . 447
4.3.63 dgeqrf . . . 449
4.3.64 dgeqrfp . . . 450
4.3.65 dgeqrt . . . 451
4.3.66 dgerqf . . . 452
4.3.67 dgetrf . . . 454
4.3.68 dgetrf2 . . . 455
4.3.69 dggqrf . . . 456
4.3.70 dggrqf . . . 458
4.3.71 dgttrf . . . 461
4.3.72 dorglq . . . 462
4.3.73 dorgql . . . 463
4.3.74 dorgqr . . . 465
4.3.75 dorgrq . . . 466
4.3.76 dormlq . . . 468
4.3.77 dormql . . . 470
4.3.78 dormqr . . . 472
4.3.79 dormrq . . . 473
4.3.80 dormrz . . . 475
4.3.81 dpbtrf . . . 477
4.3.82 dpftrf . . . 479
4.3.83 dpotrf . . . 480
4.3.84 dpotrf2 . . . 481
4.3.85 dpptrf . . . 482
4.3.86 dpstrf . . . 483
4.3.87 dpttrf . . . 485
4.3.88 dsptrf . . . 486
4.3.89 dsytrf . . . 487
4.3.90 dsytrf_aa . . . 488
4.3.91 dsytrf_rk . . . 490
4.3.92 dsytrf_rook . . . 492
4.3.93 dtplqt . . . 494
4.3.94 dtpmlqt . . . 495
4.3.95 dtpmqrt . . . 497
4.3.96 dtpqrt . . . 499
4.3.97 dtzrzf . . . 501
4.3.98 sgbtrf . . . 502
4.3.99 sgelq . . . 504
4.3.100 sgelqf . . . 505
4.3.101 sgelqt . . . 507
4.3.102 sgemlq . . . 508
4.3.103 sgemlqt . . . 510
4.3.104 sgemqr . . . 512
4.3.105 sgemqrt . . . 513
4.3.106 sgeqlf . . . 515
4.3.107 sgeqp3 . . . 517
4.3.108 sgeqr . . . 518
4.3.109 sgeqrf . . . 520

4.3.110 sgeqrfp . . . 521
4.3.111 sgeqrt . . . 522
4.3.112 sgerqf . . . 523
4.3.113 sgetrf . . . 525
4.3.114 sgetrf2 . . . 526
4.3.115 sggqrf . . . 527
4.3.116 sggrqf . . . 529
4.3.117 sgttrf . . . 532
4.3.118 sorglq . . . 533
4.3.119 sorgql . . . 534
4.3.120 sorgqr . . . 536
4.3.121 sorgrq . . . 537
4.3.122 sormlq . . . 539
4.3.123 sormql . . . 541
4.3.124 sormqr . . . 543
4.3.125 sormrq . . . 544
4.3.126 sormrz . . . 546
4.3.127 spbtrf . . . 548
4.3.128 spftrf . . . 550
4.3.129 spotrf . . . 551
4.3.130 spotrf2 . . . 552
4.3.131 spptrf . . . 553
4.3.132 spstrf . . . 554
4.3.133 spttrf . . . 556
4.3.134 ssptrf . . . 557
4.3.135 ssytrf . . . 558
4.3.136 ssytrf_aa . . . 559
4.3.137 ssytrf_rk . . . 561
4.3.138 ssytrf_rook . . . 563
4.3.139 stplqt . . . 565
4.3.140 stpmlqt . . . 566
4.3.141 stpmqrt . . . 568
4.3.142 stpqrt . . . 570
4.3.143 stzrzf . . . 572
4.3.144 zgbtrf . . . 573
4.3.145 zgelq . . . 575
4.3.146 zgelqf . . . 576
4.3.147 zgelqt . . . 578
4.3.148 zgemlq . . . 579
4.3.149 zgemlqt . . . 581
4.3.150 zgemqr . . . 583
4.3.151 zgemqrt . . . 585
4.3.152 zgeqlf . . . 587
4.3.153 zgeqp3 . . . 588
4.3.154 zgeqr . . . 589
4.3.155 zgeqrf . . . 591
4.3.156 zgeqrfp . . . 592
4.3.157 zgeqrt . . . 593
4.3.158 zgerqf . . . 595
4.3.159 zgetrf . . . 596
4.3.160 zgetrf2 . . . 597
4.3.161 zggqrf . . . 599
4.3.162 zggrqf . . . 601
4.3.163 zgttrf . . . 603
4.3.164 zhetrf . . . 604
4.3.165 zhetrf_aa . . . 606
4.3.166 zhetrf_rk . . . 607
4.3.167 zhetrf_rook . . . 610

4.3.168 zhptrf . . . 612
4.3.169 zpbtrf . . . 613
4.3.170 zpftrf . . . 614
4.3.171 zpotrf . . . 616
4.3.172 zpotrf2 . . . 617
4.3.173 zpptrf . . . 618
4.3.174 zpstrf . . . 619
4.3.175 zpttrf . . . 621
4.3.176 zsptrf . . . 622
4.3.177 zsytrf . . . 623
4.3.178 zsytrf_aa . . . 625
4.3.179 zsytrf_rk . . . 626
4.3.180 zsytrf_rook . . . 629
4.3.181 ztplqt . . . 630
4.3.182 ztpmlqt . . . 632
4.3.183 ztpmqrt . . . 634
4.3.184 ztpqrt . . . 636
4.3.185 ztzrzf . . . 638
4.3.186 zunglq . . . 639
4.3.187 zungql . . . 640
4.3.188 zungqr . . . 642
4.3.189 zungrq . . . 643
4.3.190 zunmlq . . . 645
4.3.191 zunmql . . . 647
4.3.192 zunmqr . . . 649
4.3.193 zunmrq . . . 651
4.3.194 zunmrz . . . 652
4.4 LAPACK matrix inversion routines . . . 654
4.4.1 cgetri . . . 654
4.4.2 chetri . . . 656
4.4.3 chetri2 . . . 657
4.4.4 chetri2x . . . 658
4.4.5 chetri_3 . . . 660
4.4.6 chetri_rook . . . 661
4.4.7 chptri . . . 662
4.4.8 cpftri . . . 664
4.4.9 cpotri . . . 665
4.4.10 cpptri . . . 666
4.4.11 csptri . . . 667
4.4.12 csytri . . . 668
4.4.13 csytri2 . . . 669
4.4.14 csytri2x . . . 670
4.4.15 csytri_3 . . . 671
4.4.16 csytri_rook . . . 673
4.4.17 ctftri . . . 674
4.4.18 ctptri . . . 676
4.4.19 ctrtri . . . 677
4.4.20 dgetri . . . 678
4.4.21 dpftri . . . 679
4.4.22 dpotri . . . 680
4.4.23 dpptri . . . 681
4.4.24 dsptri . . . 682
4.4.25 dsytri . . . 683
4.4.26 dsytri2 . . . 684
4.4.27 dsytri2x . . . 686
4.4.28 dsytri_3 . . . 687
4.4.29 dsytri_rook . . . 689
4.4.30 dtftri . . . 690

4.4.31 dtptri . . . 691
4.4.32 dtrtri . . . 692
4.4.33 sgetri . . . 693
4.4.34 spftri . . . 694
4.4.35 spotri . . . 695
4.4.36 spptri . . . 696
4.4.37 ssptri . . . 697
4.4.38 ssytri . . . 698
4.4.39 ssytri2 . . . 700
4.4.40 ssytri2x . . . 701
4.4.41 ssytri_3 . . . 702
4.4.42 ssytri_rook . . . 704
4.4.43 stftri . . . 705
4.4.44 stptri . . . 706
4.4.45 strtri . . . 707
4.4.46 zgetri . . . 708
4.4.47 zhetri . . . 710
4.4.48 zhetri2 . . . 711
4.4.49 zhetri2x . . . 712
4.4.50 zhetri_3 . . . 713
4.4.51 zhetri_rook . . . 715
4.4.52 zhptri . . . 716
4.4.53 zpftri . . . 717
4.4.54 zpotri . . . 719
4.4.55 zpptri . . . 720
4.4.56 zsptri . . . 720
4.4.57 zsytri . . . 722
4.4.58 zsytri2 . . . 723
4.4.59 zsytri2x . . . 724
4.4.60 zsytri_3 . . . 725
4.4.61 zsytri_rook . . . 727
4.4.62 ztftri . . . 728
4.4.63 ztptri . . . 730
4.4.64 ztrtri . . . 731
4.5 LAPACK least squares routines . . . 732
4.5.1 cgels . . . 732
4.5.2 cgelsd . . . 734
4.5.3 cgelss . . . 736
4.5.4 cgelsy . . . 738
4.5.5 cggglm . . . 741
4.5.6 cgglse . . . 743
4.5.7 dgels . . . 745
4.5.8 dgelsd . . . 747
4.5.9 dgelss . . . 750
4.5.10 dgelsy . . . 752
4.5.11 dggglm . . . 754
4.5.12 dgglse . . . 756
4.5.13 sgels . . . 758
4.5.14 sgelsd . . . 760
4.5.15 sgelss . . . 763
4.5.16 sgelsy . . . 765
4.5.17 sggglm . . . 767
4.5.18 sgglse . . . 769
4.5.19 zgels . . . 771
4.5.20 zgelsd . . . 773
4.5.21 zgelss . . . 776
4.5.22 zgelsy . . . 778
4.5.23 zggglm . . . 780

4.5.24 zgglse . . . 783
4.6 LAPACK linear equation solving routines . . . 785
4.6.1 cgbsv . . . 785
4.6.2 cgbsvx . . . 786
4.6.3 cgbsvxx . . . 791
4.6.4 cgbtrs . . . 797
4.6.5 cgesv . . . 798
4.6.6 cgesvx . . . 800
4.6.7 cgesvxx . . . 804
4.6.8 cgetrs . . . 810
4.6.9 cgtsv . . . 811
4.6.10 cgtsvx . . . 812
4.6.11 cgttrs . . . 816
4.6.12 chesv . . . 817
4.6.13 chesv_aa . . . 819
4.6.14 chesv_rk . . . 821
4.6.15 chesv_rook . . . 824
4.6.16 chesvx . . . 826
4.6.17 chesvxx . . . 829
4.6.18 chetrs . . . 835
4.6.19 chetrs2 . . . 836
4.6.20 chetrs_3 . . . 837
4.6.21 chetrs_aa . . . 839
4.6.22 chetrs_rook . . . 840
4.6.23 chpsv . . . 842
4.6.24 chpsvx . . . 843
4.6.25 cpbsv . . . 846
4.6.26 cpbsvx . . . 848
4.6.27 cpbtrs . . . 852
4.6.28 cpftrs . . . 853
4.6.29 cposv . . . 854
4.6.30 cposvx . . . 856
4.6.31 cposvxx . . . 859
4.6.32 cpotrs . . . 865
4.6.33 cppsv . . . 866
4.6.34 cppsvx . . . 868
4.6.35 cpptrs . . . 871
4.6.36 cptsv . . . 872
4.6.37 cptsvx . . . 873
4.6.38 cpttrs . . . 876
4.6.39 cspsv . . . 877
4.6.40 cspsvx . . . 879
4.6.41 csptrs . . . 882
4.6.42 csysv . . . 883
4.6.43 csysv_aa . . . 885
4.6.44 csysv_rk . . . 887
4.6.45 csysv_rook . . . 889
4.6.46 csysvx . . . 891
4.6.47 csysvxx . . . 895
4.6.48 csytrs . . . 900
4.6.49 csytrs2 . . . 902
4.6.50 csytrs_3 . . . 903
4.6.51 csytrs_aa . . . 905
4.6.52 csytrs_rook . . . 906
4.6.53 ctbtrs . . . 907
4.6.54 ctptrs . . . 909
4.6.55 dgbsv . . . 911
4.6.56 dgbsvx . . . 912

4.6.57 dgbsvxx . . . 916
4.6.58 dgbtrs . . . 923
4.6.59 dgesv . . . 924
4.6.60 dgesvx . . . 926
4.6.61 dgesvxx . . . 929
4.6.62 dgetrs . . . 936
4.6.63 dgtsv . . . 937
4.6.64 dgtsvx . . . 938
4.6.65 dgttrs . . . 942
4.6.66 dpbsv . . . 943
4.6.67 dpbsvx . . . 945
4.6.68 dpbtrs . . . 948
4.6.69 dpftrs . . . 950
4.6.70 dposv . . . 951
4.6.71 dposvx . . . 952
4.6.72 dposvxx . . . 956
4.6.73 dpotrs . . . 961
4.6.74 dppsv . . . 963
4.6.75 dppsvx . . . 964
4.6.76 dpptrs . . . 967
4.6.77 dptsv . . . 969
4.6.78 dptsvx . . . 970
4.6.79 dpttrs . . . 972
4.6.80 dsgesv . . . 974
4.6.81 dsposv . . . 976
4.6.82 dspsv . . . 978
4.6.83 dspsvx . . . 980
4.6.84 dsptrs . . . 983
4.6.85 dsysv . . . 984
4.6.86 dsysv_aa . . . 986
4.6.87 dsysv_rk . . . 988
4.6.88 dsysv_rook . . . 990
4.6.89 dsysvx . . . 992
4.6.90 dsysvxx . . . 996
4.6.91 dsytrs . . . 1001
4.6.92 dsytrs2 . . . 1002
4.6.93 dsytrs_3 . . . 1004
4.6.94 dsytrs_aa . . . 1005
4.6.95 dsytrs_rook . . . 1007
4.6.96 dtbtrs . . . 1008
4.6.97 dtptrs . . . 1010
4.6.98 sgbsv . . . 1011
4.6.99 sgbsvx . . . 1013
4.6.100 sgbsvxx . . . 1017
4.6.101 sgbtrs . . . 1023
4.6.102 sgesv . . . 1025
4.6.103 sgesvx . . . 1026
4.6.104 sgesvxx . . . 1030
4.6.105 sgetrs . . . 1036
4.6.106 sgtsv . . . 1037
4.6.107 sgtsvx . . . 1039
4.6.108 sgttrs . . . 1042
4.6.109 spbsv . . . 1044
4.6.110 spbsvx . . . 1045
4.6.111 spbtrs . . . 1049
4.6.112 spftrs . . . 1050
4.6.113 sposv . . . 1052
4.6.114 sposvx . . . 1053

4.6.115 sposvxx . . . 1057
4.6.116 spotrs . . . 1062
4.6.117 sppsv . . . 1063
4.6.118 sppsvx . . . 1065
4.6.119 spptrs . . . 1068
4.6.120 sptsv . . . 1069
4.6.121 sptsvx . . . 1071
4.6.122 spttrs . . . 1073
4.6.123 sspsv . . . 1074
4.6.124 sspsvx . . . 1076
4.6.125 ssptrs . . . 1079
4.6.126 ssysv . . . 1080
4.6.127 ssysv_aa . . . 1082
4.6.128 ssysv_rk . . . 1084
4.6.129 ssysv_rook . . . 1086
4.6.130 ssysvx . . . 1088
4.6.131 ssysvxx . . . 1092
4.6.132 ssytrs . . . 1097
4.6.133 ssytrs2 . . . 1099
4.6.134 ssytrs_3 . . . 1100
4.6.135 ssytrs_aa . . . 1102
4.6.136 ssytrs_rook . . . 1103
4.6.137 stbtrs . . . 1104
4.6.138 stptrs . . . 1106
4.6.139 zcgesv . . . 1107
4.6.140 zcposv . . . 1110
4.6.141 zgbsv . . . 1112
4.6.142 zgbsvx . . . 1114
4.6.143 zgbsvxx . . . 1118
4.6.144 zgbtrs . . . 1124
4.6.145 zgesv . . . 1126
4.6.146 zgesvx . . . 1127
4.6.147 zgesvxx . . . 1131
4.6.148 zgetrs . . . 1137
4.6.149 zgtsv . . . 1139
4.6.150 zgtsvx . . . 1140
4.6.151 zgttrs . . . 1144
4.6.152 zhesv . . . 1145
4.6.153 zhesv_aa . . . 1147
4.6.154 zhesv_rk . . . 1149
4.6.155 zhesv_rook . . . 1151
4.6.156 zhesvx . . . 1153
4.6.157 zhesvxx . . . 1157
4.6.158 zhetrs . . . 1162
4.6.159 zhetrs2 . . . 1164
4.6.160 zhetrs_3 . . . 1165
4.6.161 zhetrs_aa . . . 1167
4.6.162 zhetrs_rook . . . 1168
4.6.163 zhpsv . . . 1169
4.6.164 zhpsvx . . . 1171
4.6.165 zpbsv . . . 1174
4.6.166 zpbsvx . . . 1175
4.6.167 zpbtrs . . . 1179
4.6.168 zpftrs . . . 1180
4.6.169 zposv . . . 1182
4.6.170 zposvx . . . 1183
4.6.171 zposvxx . . . 1187
4.6.172 zpotrs . . . 1192

4.6.173 zppsv 1194
4.6.174 zppsvx 1195
4.6.175 zpptrs 1198
4.6.176 zptsv 1200
4.6.177 zptsvx 1201
4.6.178 zpttrs 1204
4.6.179 zspsv 1205
4.6.180 zspsvx 1207
4.6.181 zsptrs 1210
4.6.182 zsysv 1211
4.6.183 zsysv_aa 1213
4.6.184 zsysv_rk 1215
4.6.185 zsysv_rook 1217
4.6.186 zsysvx 1219
4.6.187 zsysvxx 1222
4.6.188 zsytrs 1228
4.6.189 zsytrs2 1229
4.6.190 zsytrs_3 1231
4.6.191 zsytrs_aa 1233
4.6.192 zsytrs_rook 1234
4.6.193 ztbtrs 1235
4.6.194 ztptrs 1237
4.7 LAPACK matrix equilibration routines 1238
4.7.1 cgbequ 1238
4.7.2 cgbequb 1240
4.7.3 cgeequ 1242
4.7.4 cgeequb 1243
4.7.5 cheequb 1245
4.7.6 cpbequ 1246
4.7.7 cpoequ 1248
4.7.8 cpoequb 1249
4.7.9 cppequ 1250
4.7.10 csyequb 1251
4.7.11 dgbequ 1253
4.7.12 dgbequb 1254
4.7.13 dgeequ 1256
4.7.14 dgeequb 1258
4.7.15 dpbequ 1259
4.7.16 dpoequ 1261
4.7.17 dpoequb 1262
4.7.18 dppequ 1263
4.7.19 dsyequb 1264
4.7.20 sgbequ 1266
4.7.21 sgbequb 1267
4.7.22 sgeequ 1269
4.7.23 sgeequb 1270
4.7.24 spbequ 1272
4.7.25 spoequ 1273
4.7.26 spoequb 1275
4.7.27 sppequ 1276
4.7.28 ssyequb 1277
4.7.29 zgbequ 1278
4.7.30 zgbequb 1280
4.7.31 zgeequ 1282
4.7.32 zgeequb 1283
4.7.33 zheequb 1285
4.7.34 zpbequ 1286
4.7.35 zpoequ 1288

4.7.36 zpoequb 1289
4.7.37 zppequ 1290
4.7.38 zsyequb 1291
4.8 LAPACK cosine sine routines 1293
4.8.1 cbbcsd 1293
4.8.2 cunbdb 1297
4.8.3 cuncsd 1300
4.8.4 cuncsd2by1 1303
4.8.5 dbbcsd 1306
4.8.6 dorbdb 1310
4.8.7 dorcsd 1313
4.8.8 dorcsd2by1 1317
4.8.9 sbbcsd 1319
4.8.10 sorbdb 1323
4.8.11 sorcsd 1326
4.8.12 sorcsd2by1 1329
4.8.13 zbbcsd 1332
4.8.14 zunbdb 1336
4.8.15 zuncsd 1339
4.8.16 zuncsd2by1 1342
4.9 LAPACK solution refinement routines 1345
4.9.1 cgbrfs 1345
4.9.2 cgbrfsx 1348
4.9.3 cgerfs 1353
4.9.4 cgerfsx 1355
4.9.5 cgtrfs 1359
4.9.6 cherfs 1362
4.9.7 cherfsx 1364
4.9.8 chprfs 1368
4.9.9 cpbrfs 1370
4.9.10 cporfs 1372
4.9.11 cporfsx 1374
4.9.12 cpprfs 1379
4.9.13 cptrfs 1380
4.9.14 csprfs 1382
4.9.15 csyrfs 1384
4.9.16 csyrfsx 1387
4.9.17 ctbrfs 1391
4.9.18 ctprfs 1393
4.9.19 ctrrfs 1395
4.9.20 dgbrfs 1397
4.9.21 dgbrfsx 1400
4.9.22 dgerfs 1404
4.9.23 dgerfsx 1406
4.9.24 dgtrfs 1411
4.9.25 dpbrfs 1413
4.9.26 dporfs 1415
4.9.27 dporfsx 1417
4.9.28 dpprfs 1422
4.9.29 dptrfs 1424
4.9.30 dsprfs 1425
4.9.31 dsyrfs 1427
4.9.32 dsyrfsx 1430
4.9.33 dtbrfs 1434
4.9.34 dtprfs 1436
4.9.35 dtrrfs 1438
4.9.36 sgbrfs 1440
4.9.37 sgbrfsx 1442

4.9.38 sgerfs . . . 1447
4.9.39 sgerfsx . . . 1449
4.9.40 sgtrfs . . . 1453
4.9.41 spbrfs . . . 1456
4.9.42 sporfs . . . 1458
4.9.43 sporfsx . . . 1460
4.9.44 spprfs . . . 1464
4.9.45 sptrfs . . . 1466
4.9.46 ssprfs . . . 1468
4.9.47 ssyrfs . . . 1470
4.9.48 ssyrfsx . . . 1472
4.9.49 stbrfs . . . 1476
4.9.50 stprfs . . . 1478
4.9.51 strrfs . . . 1480
4.9.52 zgbrfs . . . 1482
4.9.53 zgbrfsx . . . 1485
4.9.54 zgerfs . . . 1490
4.9.55 zgerfsx . . . 1492
4.9.56 zgtrfs . . . 1496
4.9.57 zherfs . . . 1499
4.9.58 zherfsx . . . 1501
4.9.59 zhprfs . . . 1505
4.9.60 zpbrfs . . . 1507
4.9.61 zporfs . . . 1509
4.9.62 zporfsx . . . 1511
4.9.63 zpprfs . . . 1516
4.9.64 zptrfs . . . 1517
4.9.65 zsprfs . . . 1519
4.9.66 zsyrfs . . . 1521
4.9.67 zsyrfsx . . . 1524
4.9.68 ztbrfs . . . 1528
4.9.69 ztprfs . . . 1530
4.9.70 ztrrfs . . . 1532
4.10 LAPACK singular value decompositions routines . . . 1534
4.10.1 cbdsqr . . . 1534
4.10.2 cgbbrd . . . 1537
4.10.3 cgebrd . . . 1539
4.10.4 cgejsv . . . 1541
4.10.5 cgesdd . . . 1546
4.10.6 cgesvd . . . 1548
4.10.7 cgesvdx . . . 1550
4.10.8 cgesvj . . . 1554
4.10.9 cggsvd3 . . . 1557
4.10.10 cggsvp3 . . . 1561
4.10.11 ctgsja . . . 1564
4.10.12 cungbr . . . 1568
4.10.13 cunmbr . . . 1570
4.10.14 dbdsdc . . . 1572
4.10.15 dbdsqr . . . 1574
4.10.16 dbdsvdx . . . 1577
4.10.17 dgbbrd . . . 1579
4.10.18 dgebrd . . . 1581
4.10.19 dgejsv . . . 1583
4.10.20 dgesdd . . . 1587
4.10.21 dgesvd . . . 1589
4.10.22 dgesvdx . . . 1592
4.10.23 dgesvj . . . 1595
4.10.24 dggsvd3 . . . 1597

4.10.25 dggsvp3 . . . 1602
4.10.26 dorgbr . . . 1605
4.10.27 dormbr . . . 1606
4.10.28 dtgsja . . . 1609
4.10.29 sbdsdc . . . 1613
4.10.30 sbdsqr . . . 1615
4.10.31 sbdsvdx . . . 1617
4.10.32 sgbbrd . . . 1620
4.10.33 sgebrd . . . 1622
4.10.34 sgejsv . . . 1623
4.10.35 sgesdd . . . 1628
4.10.36 sgesvd . . . 1630
4.10.37 sgesvdx . . . 1632
4.10.38 sgesvj . . . 1635
4.10.39 sggsvd3 . . . 1638
4.10.40 sggsvp3 . . . 1642
4.10.41 sorgbr . . . 1645
4.10.42 sormbr . . . 1647
4.10.43 stgsja . . . 1649
4.10.44 zbdsqr . . . 1653
4.10.45 zgbbrd . . . 1655
4.10.46 zgebrd . . . 1657
4.10.47 zgejsv . . . 1659
4.10.48 zgesdd . . . 1664
4.10.49 zgesvd . . . 1667
4.10.50 zgesvdx . . . 1669
4.10.51 zgesvj . . . 1672
4.10.52 zggsvd3 . . . 1675
4.10.53 zggsvp3 . . . 1679
4.10.54 ztgsja . . . 1682
4.10.55 zungbr . . . 1687
4.10.56 zunmbr . . . 1688
4.11 LAPACK symmetric eigenvalues routines . . . 1690
4.11.1 chbev . . . 1690
4.11.2 chbevd . . . 1692
4.11.3 chbevx . . . 1694
4.11.4 chbtrd . . . 1697
4.11.5 cheev . . . 1699
4.11.6 cheevd . . . 1701
4.11.7 cheevr . . . 1703
4.11.8 cheevx . . . 1707
4.11.9 chetrd . . . 1709
4.11.10 chpev . . . 1711
4.11.11 chpevd . . . 1712
4.11.12 chpevx . . . 1715
4.11.13 chptrd . . . 1717
4.11.14 cpteqr . . . 1719
4.11.15 cstedc . . . 1720
4.11.16 cstegr . . . 1722
4.11.17 cstein . . . 1725
4.11.18 cstemr . . . 1727
4.11.19 csteqr . . . 1731
4.11.20 cungtr . . . 1732
4.11.21 cunm22 . . . 1733
4.11.22 cunmtr . . . 1735
4.11.23 cupgtr . . . 1737
4.11.24 cupmtr . . . 1738
4.11.25 ddisna . . . 1740

4.11.26 dopgtr 1741
4.11.27 dopmtr 1743
4.11.28 dorgtr 1744
4.11.29 dorm22 1746
4.11.30 dormtr 1747
4.11.31 dpteqr 1749
4.11.32 dsbev 1751
4.11.33 dsbevd 1752
4.11.34 dsbevx 1754
4.11.35 dsbtrd 1757
4.11.36 dspev 1759
4.11.37 dspevd 1760
4.11.38 dspevx 1762
4.11.39 dsptrd 1765
4.11.40 dstebz 1766
4.11.41 dstedc 1769
4.11.42 dstegr 1770
4.11.43 dstein 1773
4.11.44 dstemr 1775
4.11.45 dsteqr 1779
4.11.46 dsterf 1780
4.11.47 dstev 1781
4.11.48 dstevd 1782
4.11.49 dstevr 1784
4.11.50 dstevx 1787
4.11.51 dsyev 1790
4.11.52 dsyevd 1791
4.11.53 dsyevr 1793
4.11.54 dsyevx 1796
4.11.55 dsytrd 1799
4.11.56 sdisna 1801
4.11.57 sopgtr 1802
4.11.58 sopmtr 1803
4.11.59 sorgtr 1805
4.11.60 sorm22 1806
4.11.61 sormtr 1808
4.11.62 spteqr 1810
4.11.63 ssbev 1812
4.11.64 ssbevd 1813
4.11.65 ssbevx 1815
4.11.66 ssbtrd 1818
4.11.67 sspev 1820
4.11.68 sspevd 1821
4.11.69 sspevx 1823
4.11.70 ssptrd 1826
4.11.71 sstebz 1827
4.11.72 sstedc 1829
4.11.73 sstegr 1831
4.11.74 sstein 1834
4.11.75 sstemr 1836
4.11.76 ssteqr 1839
4.11.77 ssterf 1841
4.11.78 sstev 1842
4.11.79 sstevd 1843
4.11.80 sstevr 1845
4.11.81 sstevx 1848
4.11.82 ssyev 1850
4.11.83 ssyevd 1852

4.11.84 ssyevr . . . 1853
4.11.85 ssyevx . . . 1857
4.11.86 ssytrd . . . 1860
4.11.87 zhbev . . . 1862
4.11.88 zhbevd . . . 1863
4.11.89 zhbevx . . . 1866
4.11.90 zhbtrd . . . 1869
4.11.91 zheev . . . 1870
4.11.92 zheevd . . . 1872
4.11.93 zheevr . . . 1874
4.11.94 zheevx . . . 1878
4.11.95 zhetrd . . . 1881
4.11.96 zhpev . . . 1882
4.11.97 zhpevd . . . 1884
4.11.98 zhpevx . . . 1886
4.11.99 zhptrd . . . 1888
4.11.100 zpteqr . . . 1890
4.11.101 zstedc . . . 1891
4.11.102 zstegr . . . 1893
4.11.103 zstein . . . 1896
4.11.104 zstemr . . . 1898
4.11.105 zsteqr . . . 1902
4.11.106 zungtr . . . 1903
4.11.107 zunm22 . . . 1904
4.11.108 zunmtr . . . 1906
4.11.109 zupgtr . . . 1908
4.11.110 zupmtr . . . 1910
4.12 LAPACK generalized symmetric eigenvalues routines . . . 1911
4.12.1 chbgst . . . 1911
4.12.2 chbgv . . . 1913
4.12.3 chbgvd . . . 1915
4.12.4 chbgvx . . . 1918
4.12.5 chegst . . . 1921
4.12.6 chegv . . . 1923
4.12.7 chegvd . . . 1925
4.12.8 chegvx . . . 1927
4.12.9 chpgst . . . 1931
4.12.10 chpgv . . . 1932
4.12.11 chpgvd . . . 1934
4.12.12 chpgvx . . . 1936
4.12.13 cpbstf . . . 1939
4.12.14 dpbstf . . . 1940
4.12.15 dsbgst . . . 1941
4.12.16 dsbgv . . . 1943
4.12.17 dsbgvd . . . 1945
4.12.18 dsbgvx . . . 1948
4.12.19 dspgst . . . 1951
4.12.20 dspgv . . . 1952
4.12.21 dspgvd . . . 1954
4.12.22 dspgvx . . . 1956
4.12.23 dsygst . . . 1959
4.12.24 dsygv . . . 1960
4.12.25 dsygvd . . . 1962
4.12.26 dsygvx . . . 1964
4.12.27 spbstf . . . 1967
4.12.28 ssbgst . . . 1969
4.12.29 ssbgv . . . 1970
4.12.30 ssbgvd . . . 1972

4.12.31 ssbgvx 1975
4.12.32 sspgst 1978
4.12.33 sspgv 1979
4.12.34 sspgvd 1981
4.12.35 sspgvx 1983
4.12.36 ssygst 1986
4.12.37 ssygv 1987
4.12.38 ssygvd 1989
4.12.39 ssygvx 1991
4.12.40 zhbgst 1994
4.12.41 zhbgv 1996
4.12.42 zhbgvd 1998
4.12.43 zhbgvx 2001
4.12.44 zhegst 2004
4.12.45 zhegv 2006
4.12.46 zhegvd 2008
4.12.47 zhegvx 2010
4.12.48 zhpgst 2014
4.12.49 zhpgv 2015
4.12.50 zhpgvd 2017
4.12.51 zhpgvx 2019
4.12.52 zpbstf 2022
4.13 LAPACK non symmetric eigenvalues routines 2023
4.13.1 cgebak 2023
4.13.2 cgebal 2025
4.13.3 cgees 2026
4.13.4 cgeesx 2028
4.13.5 cgeev 2031
4.13.6 cgeevx 2033
4.13.7 cgehrd 2036
4.13.8 chsein 2038
4.13.9 chseqr 2040
4.13.10 ctrevc 2043
4.13.11 ctrexc 2045
4.13.12 ctrsen 2047
4.13.13 ctrsna 2049
4.13.14 ctrsyl 2051
4.13.15 cunghr 2053
4.13.16 cunmhr 2054
4.13.17 dgebak 2056
4.13.18 dgebal 2058
4.13.19 dgees 2059
4.13.20 dgeesx 2061
4.13.21 dgeev 2064
4.13.22 dgeevx 2066
4.13.23 dgehrd 2070
4.13.24 dhsein 2071
4.13.25 dhseqr 2074
4.13.26 dorghr 2077
4.13.27 dormhr 2078
4.13.28 dtrevc 2080
4.13.29 dtrexc 2082
4.13.30 dtrsen 2084
4.13.31 dtrsna 2087
4.13.32 dtrsyl 2089
4.13.33 sgebak 2091
4.13.34 sgebal 2092
4.13.35 sgees 2094

4.13.36 sgeesx . . . 2096
4.13.37 sgeev . . . 2099
4.13.38 sgeevx . . . 2101
4.13.39 sgehrd . . . 2104
4.13.40 shsein . . . 2106
4.13.41 shseqr . . . 2109
4.13.42 sorghr . . . 2111
4.13.43 sormhr . . . 2113
4.13.44 strevc . . . 2115
4.13.45 strexc . . . 2117
4.13.46 strsen . . . 2118
4.13.47 strsna . . . 2121
4.13.48 strsyl . . . 2124
4.13.49 zgebak . . . 2125
4.13.50 zgebal . . . 2127
4.13.51 zgees . . . 2128
4.13.52 zgeesx . . . 2130
4.13.53 zgeev . . . 2133
4.13.54 zgeevx . . . 2135
4.13.55 zgehrd . . . 2138
4.13.56 zhsein . . . 2140
4.13.57 zhseqr . . . 2143
4.13.58 ztrevc . . . 2145
4.13.59 ztrexc . . . 2147
4.13.60 ztrsen . . . 2149
4.13.61 ztrsna . . . 2151
4.13.62 ztrsyl . . . 2153
4.13.63 zunghr . . . 2155
4.13.64 zunmhr . . . 2156
4.14 LAPACK generalized non symmetric eigenvalues routines . . . 2158
4.14.1 cggbak . . . 2158
4.14.2 cggbal . . . 2160
4.14.3 cgges . . . 2162
4.14.4 cgges3 . . . 2165
4.14.5 cggesx . . . 2168
4.14.6 cggev . . . 2172
4.14.7 cggev3 . . . 2174
4.14.8 cggevx . . . 2177
4.14.9 cgghd3 . . . 2180
4.14.10 cgghrd . . . 2183
4.14.11 chgeqz . . . 2185
4.14.12 ctgevc . . . 2189
4.14.13 ctgexc . . . 2191
4.14.14 ctgsen . . . 2193
4.14.15 ctgsna . . . 2197
4.14.16 ctgsyl . . . 2199
4.14.17 dggbak . . . 2202
4.14.18 dggbal . . . 2204
4.14.19 dgges . . . 2206
4.14.20 dgges3 . . . 2209
4.14.21 dggesx . . . 2212
4.14.22 dggev . . . 2216
4.14.23 dggev3 . . . 2218
4.14.24 dggevx . . . 2221
4.14.25 dgghd3 . . . 2225
4.14.26 dgghrd . . . 2228
4.14.27 dhgeqz . . . 2230
4.14.28 dtgevc . . . 2233

4.14.29 dtgexc . . . 2236
4.14.30 dtgsen . . . 2238
4.14.31 dtgsna . . . 2242
4.14.32 dtgsyl . . . 2245
4.14.33 sggbak . . . 2248
4.14.34 sggbal . . . 2249
4.14.35 sgges . . . 2251
4.14.36 sgges3 . . . 2254
4.14.37 sggesx . . . 2257
4.14.38 sggev . . . 2261
4.14.39 sggev3 . . . 2264
4.14.40 sggevx . . . 2266
4.14.41 sgghd3 . . . 2270
4.14.42 sgghrd . . . 2273
4.14.43 shgeqz . . . 2275
4.14.44 stgevc . . . 2279
4.14.45 stgexc . . . 2281
4.14.46 stgsen . . . 2284
4.14.47 stgsna . . . 2287
4.14.48 stgsyl . . . 2290
4.14.49 zggbak . . . 2293
4.14.50 zggbal . . . 2294
4.14.51 zgges . . . 2296
4.14.52 zgges3 . . . 2299
4.14.53 zggesx . . . 2302
4.14.54 zggev . . . 2306
4.14.55 zggev3 . . . 2308
4.14.56 zggevx . . . 2311
4.14.57 zgghd3 . . . 2315
4.14.58 zgghrd . . . 2318
4.14.59 zhgeqz . . . 2320
4.14.60 ztgevc . . . 2323
4.14.61 ztgexc . . . 2326
4.14.62 ztgsen . . . 2328
4.14.63 ztgsna . . . 2331
4.14.64 ztgsyl . . . 2334
4.15 LAPACK condition number estimation routines . . . 2337
4.15.1 cgbcon . . . 2337
4.15.2 cgecon . . . 2338
4.15.3 cgtcon . . . 2340
4.15.4 checon . . . 2341
4.15.5 checon_3 . . . 2343
4.15.6 checon_rook . . . 2344
4.15.7 chpcon . . . 2346
4.15.8 cpbcon . . . 2347
4.15.9 cpocon . . . 2348
4.15.10 cppcon . . . 2350
4.15.11 cptcon . . . 2351
4.15.12 cspcon . . . 2352
4.15.13 csycon . . . 2354
4.15.14 csycon_3 . . . 2355
4.15.15 csycon_rook . . . 2357
4.15.16 ctbcon . . . 2358
4.15.17 ctpcon . . . 2360
4.15.18 ctrcon . . . 2361
4.15.19 dgbcon . . . 2363
4.15.20 dgecon . . . 2364
4.15.21 dgtcon . . . 2366

4.15.22 dpbcon 2367
4.15.23 dpocon 2369
4.15.24 dppcon 2370
4.15.25 dptcon 2371
4.15.26 dspcon 2372
4.15.27 dsycon 2374
4.15.28 dsycon_3 2375
4.15.29 dsycon_rook 2377
4.15.30 dtbcon 2378
4.15.31 dtpcon 2380
4.15.32 dtrcon 2381
4.15.33 sgbcon 2383
4.15.34 sgecon 2385
4.15.35 sgtcon 2386
4.15.36 spbcon 2388
4.15.37 spocon 2389
4.15.38 sppcon 2390
4.15.39 sptcon 2392
4.15.40 sspcon 2393
4.15.41 ssycon 2394
4.15.42 ssycon_3 2396
4.15.43 ssycon_rook 2397
4.15.44 stbcon 2399
4.15.45 stpcon 2400
4.15.46 strcon 2402
4.15.47 zgbcon 2403
4.15.48 zgecon 2405
4.15.49 zgtcon 2406
4.15.50 zhecon 2408
4.15.51 zhecon_3 2409
4.15.52 zhecon_rook 2411
4.15.53 zhpcon 2412
4.15.54 zpbcon 2414
4.15.55 zpocon 2415
4.15.56 zppcon 2416
4.15.57 zptcon 2418
4.15.58 zspcon 2419
4.15.59 zsycon 2420
4.15.60 zsycon_3 2421
4.15.61 zsycon_rook 2423
4.15.62 ztbcon 2425
4.15.63 ztpcon 2426
4.15.64 ztrcon 2428
4.16 LAPACK lapack routines routines 2429
4.16.1 cgetsls 2429
4.16.2 chb2st_kernels 2431
4.16.3 chbev_2stage 2433
4.16.4 chbevd_2stage 2435
4.16.5 chbevx_2stage 2437
4.16.6 cheev_2stage 2440
4.16.7 cheevd_2stage 2442
4.16.8 cheevr_2stage 2444
4.16.9 cheevx_2stage 2448
4.16.10 chegv_2stage 2451
4.16.11 chesv_aa_2stage 2453
4.16.12 chetf2_rk 2455
4.16.13 chetrd_2stage 2457
4.16.14 chetrd_he2hb 2459

4.16.15 chetrf_aa_2stage 2461
4.16.16 chetri_3x 2463
4.16.17 chetrs_aa_2stage 2465
4.16.18 chptrs 2466
4.16.19 clahef_rk 2467
4.16.20 clasyf_rk 2470
4.16.21 csyconvf 2472
4.16.22 csyconvf_rook 2474
4.16.23 csysv_aa_2stage 2476
4.16.24 csytf2_rk 2478
4.16.25 csytrf_aa_2stage 2480
4.16.26 csytri_3x 2482
4.16.27 csytrs_aa_2stage 2484
4.16.28 ctrevc3 2485
4.16.29 ctrtrs 2488
4.16.30 dcabs1 2489
4.16.31 dgetsls 2490
4.16.32 dlasyf_rk 2492
4.16.33 dsb2st_kernels 2494
4.16.34 dsbev_2stage 2496
4.16.35 dsbevd_2stage 2498
4.16.36 dsbevx_2stage 2500
4.16.37 dsyconvf 2503
4.16.38 dsyconvf_rook 2505
4.16.39 dsyev_2stage 2507
4.16.40 dsyevd_2stage 2508
4.16.41 dsyevr_2stage 2510
4.16.42 dsyevx_2stage 2514
4.16.43 dsygv_2stage 2517
4.16.44 dsysv_aa_2stage 2519
4.16.45 dsytf2_rk 2521
4.16.46 dsytrd_2stage 2523
4.16.47 dsytrd_sy2sb 2525
4.16.48 dsytrf_aa_2stage 2527
4.16.49 dsytri_3x 2529
4.16.50 dsytrs_aa_2stage 2530
4.16.51 dtrevc3 2532
4.16.52 dtrtrs 2534
4.16.53 icmax1 2536
4.16.54 ilaclc 2537
4.16.55 ilaclr 2537
4.16.56 iladlc 2538
4.16.57 iladlr 2539
4.16.58 ilaenv2stage 2540
4.16.59 ilaslc 2541
4.16.60 ilaslr 2542
4.16.61 ilazlc 2542
4.16.62 ilazlr 2543
4.16.63 izmax1 2544
4.16.64 nancheck_flag 2545
4.16.65 scabs1 2545
4.16.66 sgetsls 2545
4.16.67 slasyf_rk 2547
4.16.68 ssb2st_kernels 2550
4.16.69 ssbev_2stage 2551
4.16.70 ssbevd_2stage 2553
4.16.71 ssbevx_2stage 2555
4.16.72 ssyconvf 2558

4.16.73 ssyconvf_rook . . . 2560
4.16.74 ssyev_2stage . . . 2562
4.16.75 ssyevd_2stage . . . 2564
4.16.76 ssyevr_2stage . . . 2566
4.16.77 ssyevx_2stage . . . 2570
4.16.78 ssygv_2stage . . . 2572
4.16.79 ssysv_aa_2stage . . . 2574
4.16.80 ssytf2_rk . . . 2577
4.16.81 ssytrd_2stage . . . 2579
4.16.82 ssytrd_sy2sb . . . 2581
4.16.83 ssytrf_aa_2stage . . . 2582
4.16.84 ssytri_3x . . . 2584
4.16.85 ssytrs_aa_2stage . . . 2586
4.16.86 strevc3 . . . 2587
4.16.87 strtrs . . . 2590
4.16.88 zgetsls . . . 2591
4.16.89 zhb2st_kernels . . . 2593
4.16.90 zhbev_2stage . . . 2595
4.16.91 zhbevd_2stage . . . 2597
4.16.92 zhbevx_2stage . . . 2599
4.16.93 zheev_2stage . . . 2602
4.16.94 zheevd_2stage . . . 2604
4.16.95 zheevr_2stage . . . 2606
4.16.96 zheevx_2stage . . . 2610
4.16.97 zhegv_2stage . . . 2613
4.16.98 zhesv_aa_2stage . . . 2615
4.16.99 zhetf2_rk . . . 2618
4.16.100 zhetrd_2stage . . . 2620
4.16.101 zhetrd_he2hb . . . 2622
4.16.102 zhetrf_aa_2stage . . . 2623
4.16.103 zhetri_3x . . . 2625
4.16.104 zhetrs_aa_2stage . . . 2627
4.16.105 zhptrs . . . 2628
4.16.106 zlahef_rk . . . 2630
4.16.107 zlasyf_rk . . . 2632
4.16.108 zsyconvf . . . 2635
4.16.109 zsyconvf_rook . . . 2637
4.16.110 zsysv_aa_2stage . . . 2639
4.16.111 zsytf2_rk . . . 2641
4.16.112 zsytrf_aa_2stage . . . 2643
4.16.113 zsytri_3x . . . 2645
4.16.114 zsytrs_aa_2stage . . . 2646
4.16.115 ztrevc3 . . . 2648
4.16.116 ztrtrs . . . 2650
4.17 LAPACK auxiliary routines . . . 2652
4.17.1 cgbtf2 . . . 2652
4.17.2 cgebd2 . . . 2653
4.17.3 cgehd2 . . . 2655
4.17.4 cgelq2 . . . 2656
4.17.5 cgelqt3 . . . 2657
4.17.6 cgeql2 . . . 2658
4.17.7 cgeqr2 . . . 2659
4.17.8 cgeqr2p . . . 2661
4.17.9 cgeqrt2 . . . 2662
4.17.10 cgeqrt3 . . . 2663
4.17.11 cgerq2 . . . 2664
4.17.12 cgesc2 . . . 2665
4.17.13 cgetc2 . . . 2666

4.17.14 cgetf2 2668
4.17.15 cgsvj0 2669
4.17.16 cgsvj1 2671
4.17.17 cgtts2 2674
4.17.18 chegs2 2675
4.17.19 cheswapr 2676
4.17.20 chetd2 2678
4.17.21 chetf2 2679
4.17.22 chetf2_rook 2680
4.17.23 chfrk 2682
4.17.24 cla_gbamv 2684
4.17.25 cla_gbrcond_c 2686
4.17.26 cla_gbrcond_x 2687
4.17.27 cla_gbrfsx_extended 2689
4.17.28 cla_gbrpvgrw 2694
4.17.29 cla_geamv 2695
4.17.30 cla_gercond_c 2697
4.17.31 cla_gercond_x 2699
4.17.32 cla_gerfsx_extended 2700
4.17.33 cla_gerpvgrw 2705
4.17.34 cla_heamv 2706
4.17.35 cla_hercond_c 2707
4.17.36 cla_hercond_x 2709
4.17.37 cla_herfsx_extended 2710
4.17.38 cla_herpvgrw 2715
4.17.39 cla_lin_berr 2716
4.17.40 cla_porcond_c 2717
4.17.41 cla_porcond_x 2719
4.17.42 cla_porfsx_extended 2720
4.17.43 cla_porpvgrw 2724
4.17.44 cla_syamv 2726
4.17.45 cla_syrcond_c 2727
4.17.46 cla_syrcond_x 2729
4.17.47 cla_syrfsx_extended 2730
4.17.48 cla_syrpvgrw 2735
4.17.49 cla_wwaddw 2736
4.17.50 clabrd 2737
4.17.51 clacgv 2739
4.17.52 clacn2 2740
4.17.53 clacon 2741
4.17.54 clacp2 2742
4.17.55 clacpy 2743
4.17.56 clacrm 2744
4.17.57 clacrt 2745
4.17.58 cladiv 2746
4.17.59 claed0 2747
4.17.60 claed7 2748
4.17.61 claed8 2751
4.17.62 claein 2754
4.17.63 claesy 2756
4.17.64 claev2 2757
4.17.65 clag2z 2758
4.17.66 clags2 2759
4.17.67 clagtm 2761
4.17.68 clahef 2763
4.17.69 clahef_aa 2765
4.17.70 clahef_rook 2766
4.17.71 clahqr 2768

4.17.72 clahr2 2770
4.17.73 claic1 2772
4.17.74 clals0 2773
4.17.75 clalsa 2776
4.17.76 clalsd 2779
4.17.77 clamswlq 2781
4.17.78 clamtsqr 2784
4.17.79 clangb 2786
4.17.80 clange 2787
4.17.81 clangt 2788
4.17.82 clanhb 2789
4.17.83 clanhe 2790
4.17.84 clanhf 2792
4.17.85 clanhp 2793
4.17.86 clanhs 2794
4.17.87 clanht 2795
4.17.88 clansb 2796
4.17.89 clansp 2798
4.17.90 clansy 2799
4.17.91 clantb 2800
4.17.92 clantp 2802
4.17.93 clantr 2803
4.17.94 clapll 2804
4.17.95 clapmr 2805
4.17.96 clapmt 2806
4.17.97 claqgb 2808
4.17.98 claqge 2809
4.17.99 claqhb 2811
4.17.100 claqhe 2812
4.17.101 claqhp 2813
4.17.102 claqp2 2814
4.17.103 claqps 2816
4.17.104 claqr0 2818
4.17.105 claqr1 2820
4.17.106 claqr2 2821
4.17.107 claqr3 2824
4.17.108 claqr4 2827
4.17.109 claqr5 2830
4.17.110 claqsb 2833
4.17.111 claqsp 2834
4.17.112 claqsy 2835
4.17.113 clar1v 2837
4.17.114 clar2v 2839
4.17.115 clarcm 2840
4.17.116 clarf 2842
4.17.117 clarfb 2843
4.17.118 clarfg 2845
4.17.119 clarfgp 2846
4.17.120 clarft 2847
4.17.121 clarfx 2849
4.17.122 clarfy 2850
4.17.123 clargv 2852
4.17.124 clarnv 2853
4.17.125 clarrv 2854
4.17.126 clarscl2 2857
4.17.127 clartg 2858
4.17.128 clartv 2859
4.17.129 clarz 2860

4.17.130 clarzb 2862
4.17.131 clarzt 2864
4.17.132 clascl 2865
4.17.133 clascl2 2867
4.17.134 claset 2868
4.17.135 clasr 2869
4.17.136 classq 2871
4.17.137 claswlq 2872
4.17.138 claswp 2874
4.17.139 clasyf 2875
4.17.140 clasyf_aa 2877
4.17.141 clasyf_rook 2878
4.17.142 clatbs 2880
4.17.143 clatdf 2882
4.17.144 clatps 2883
4.17.145 clatrd 2885
4.17.146 clatrs 2887
4.17.147 clatrz 2888
4.17.148 clatsqr 2890
4.17.149 clauu2 2891
4.17.150 clauum 2892
4.17.151 cpbtf2 2893
4.17.152 cpotf2 2894
4.17.153 cpstf2 2896
4.17.154 cptts2 2897
4.17.155 csrscl 2898
4.17.156 csyconv 2899
4.17.157 csyswapr 2900
4.17.158 csytf2 2902
4.17.159 csytf2_rook 2903
4.17.160 ctfsm 2904
4.17.161 ctfttp 2906
4.17.162 ctfttr 2907
4.17.163 ctgex2 2909
4.17.164 ctgsy2 2910
4.17.165 ctplqt2 2913
4.17.166 ctpqrt2 2915
4.17.167 ctprfb 2916
4.17.168 ctpttf 2918
4.17.169 ctpttr 2919
4.17.170 ctrti2 2921
4.17.171 ctrttf 2922
4.17.172 ctrttp 2923
4.17.173 cunbdb1 2924
4.17.174 cunbdb2 2926
4.17.175 cunbdb3 2928
4.17.176 cunbdb4 2930
4.17.177 cunbdb5 2933
4.17.178 cunbdb6 2935
4.17.179 cung2l 2936
4.17.180 cung2r 2938
4.17.181 cungl2 2939
4.17.182 cungr2 2940
4.17.183 cunm2l 2942
4.17.184 cunm2r 2943
4.17.185 cunml2 2945
4.17.186 cunmr2 2947
4.17.187 cunmr3 2949

4.17.188 dgbtf2 . . . 2951
4.17.189 dgebd2 . . . 2952
4.17.190 dgehd2 . . . 2954
4.17.191 dgelq2 . . . 2955
4.17.192 dgelqt3 . . . 2956
4.17.193 dgeql2 . . . 2957
4.17.194 dgeqr2 . . . 2958
4.17.195 dgeqr2p . . . 2959
4.17.196 dgeqrt2 . . . 2960
4.17.197 dgeqrt3 . . . 2961
4.17.198 dgerq2 . . . 2963
4.17.199 dgesc2 . . . 2964
4.17.200 dgetc2 . . . 2965
4.17.201 dgetf2 . . . 2966
4.17.202 dgsvj0 . . . 2967
4.17.203 dgsvj1 . . . 2969
4.17.204 dgtts2 . . . 2972
4.17.205 disnan . . . 2973
4.17.206 dla_gbamv . . . 2974
4.17.207 dla_gbrcond . . . 2976
4.17.208 dla_gbrfsx_extended . . . 2978
4.17.209 dla_gbrpvgrw . . . 2983
4.17.210 dla_geamv . . . 2984
4.17.211 dla_gercond . . . 2986
4.17.212 dla_gerfsx_extended . . . 2988
4.17.213 dla_gerpvgrw . . . 2992
4.17.214 dla_lin_berr . . . 2993
4.17.215 dla_porcond . . . 2994
4.17.216 dla_porfsx_extended . . . 2996
4.17.217 dla_porpvgrw . . . 3000
4.17.218 dla_syamv . . . 3001
4.17.219 dla_syrcond . . . 3003
4.17.220 dla_syrfsx_extended . . . 3004
4.17.221 dla_syrpvgrw . . . 3009
4.17.222 dla_wwaddw . . . 3010
4.17.223 dlabrd . . . 3011
4.17.224 dlacn2 . . . 3013
4.17.225 dlacon . . . 3014
4.17.226 dlacpy . . . 3015
4.17.227 dladiv . . . 3016
4.17.228 dlae2 . . . 3017
4.17.229 dlaebz . . . 3018
4.17.230 dlaed0 . . . 3021
4.17.231 dlaed1 . . . 3023
4.17.232 dlaed2 . . . 3025
4.17.233 dlaed3 . . . 3027
4.17.234 dlaed4 . . . 3029
4.17.235 dlaed5 . . . 3030
4.17.236 dlaed6 . . . 3031
4.17.237 dlaed7 . . . 3033
4.17.238 dlaed8 . . . 3036
4.17.239 dlaed9 . . . 3038
4.17.240 dlaeda . . . 3040
4.17.241 dlaein . . . 3042
4.17.242 dlaev2 . . . 3044
4.17.243 dlaexc . . . 3045
4.17.244 dlag2 . . . 3047
4.17.245 dlag2s . . . 3048

4.17.246 dlags2 3050
4.17.247 dlagtf 3051
4.17.248 dlagtm 3053
4.17.249 dlagts 3055
4.17.250 dlagv2 3056
4.17.251 dlahqr 3058
4.17.252 dlahr2 3060
4.17.253 dlaic1 3062
4.17.254 dlaisnan 3063
4.17.255 dlaln2 3064
4.17.256 dlals0 3067
4.17.257 dlalsa 3070
4.17.258 dlalsd 3073
4.17.259 dlamrg 3075
4.17.260 dlamswlq 3076
4.17.261 dlamtsqr 3078
4.17.262 dlaneg 3080
4.17.263 dlangb 3081
4.17.264 dlange 3082
4.17.265 dlangt 3083
4.17.266 dlanhs 3085
4.17.267 dlansb 3086
4.17.268 dlansf 3087
4.17.269 dlansp 3088
4.17.270 dlanst 3089
4.17.271 dlansy 3090
4.17.272 dlantb 3092
4.17.273 dlantp 3093
4.17.274 dlantr 3094
4.17.275 dlanv2 3096
4.17.276 dlapll 3097
4.17.277 dlapmr 3098
4.17.278 dlapmt 3099
4.17.279 dlapy2 3100
4.17.280 dlapy3 3101
4.17.281 dlaqgb 3102
4.17.282 dlaqge 3103
4.17.283 dlaqp2 3105
4.17.284 dlaqps 3106
4.17.285 dlaqr0 3108
4.17.286 dlaqr1 3110
4.17.287 dlaqr2 3112
4.17.288 dlaqr3 3115
4.17.289 dlaqr4 3118
4.17.290 dlaqr5 3121
4.17.291 dlaqsb 3124
4.17.292 dlaqsp 3125
4.17.293 dlaqsy 3126
4.17.294 dlaqtr 3128
4.17.295 dlar1v 3129
4.17.296 dlar2v 3132
4.17.297 dlarf 3133
4.17.298 dlarfb 3135
4.17.299 dlarfg 3137
4.17.300 dlarfgp 3138
4.17.301 dlarft 3139
4.17.302 dlarfx 3140
4.17.303 dlarfy 3142

4.17.304 dlargv 3143
4.17.305 dlarnv 3144
4.17.306 dlarra 3145
4.17.307 dlarrb 3146
4.17.308 dlarrc 3149
4.17.309 dlarrd 3150
4.17.310 dlarre 3153
4.17.311 dlarrf 3156
4.17.312 dlarrj 3158
4.17.313 dlarrk 3160
4.17.314 dlarrr 3162
4.17.315 dlarrv 3162
4.17.316 dlarscl2 3166
4.17.317 dlartg 3167
4.17.318 dlartgp 3168
4.17.319 dlartgs 3169
4.17.320 dlartv 3170
4.17.321 dlaruv 3171
4.17.322 dlarz 3172
4.17.323 dlarzb 3173
4.17.324 dlarzt 3175
4.17.325 dlas2 3177
4.17.326 dlascl 3178
4.17.327 dlascl2 3179
4.17.328 dlasd0 3180
4.17.329 dlasd1 3182
4.17.330 dlasd2 3184
4.17.331 dlasd3 3187
4.17.332 dlasd4 3190
4.17.333 dlasd5 3191
4.17.334 dlasd6 3193
4.17.335 dlasd7 3196
4.17.336 dlasd8 3199
4.17.337 dlasda 3201
4.17.338 dlasdq 3204
4.17.339 dlasdt 3206
4.17.340 dlaset 3207
4.17.341 dlasq1 3209
4.17.342 dlasq2 3210
4.17.343 dlasq3 3211
4.17.344 dlasq4 3213
4.17.345 dlasq5 3215
4.17.346 dlasq6 3216
4.17.347 dlasr 3218
4.17.348 dlasrt 3220
4.17.349 dlassq 3221
4.17.350 dlasv2 3222
4.17.351 dlaswlq 3223
4.17.352 dlaswp 3225
4.17.353 dlasy2 3226
4.17.354 dlasyf 3228
4.17.355 dlasyf_aa 3230
4.17.356 dlasyf_rook 3231
4.17.357 dlat2s 3233
4.17.358 dlatbs 3234
4.17.359 dlatdf 3236
4.17.360 dlatps 3237
4.17.361 dlatrd 3239

4.17.362 dlatrs . . . 3241
4.17.363 dlatrz . . . 3242
4.17.364 dlatsqr . . . 3243
4.17.365 dlauu2 . . . 3245
4.17.366 dlauum . . . 3246
4.17.367 dorbdb1 . . . 3247
4.17.368 dorbdb2 . . . 3249
4.17.369 dorbdb3 . . . 3251
4.17.370 dorbdb4 . . . 3253
4.17.371 dorbdb5 . . . 3256
4.17.372 dorbdb6 . . . 3257
4.17.373 dorg2l . . . 3259
4.17.374 dorg2r . . . 3261
4.17.375 dorgl2 . . . 3262
4.17.376 dorgr2 . . . 3263
4.17.377 dorm2l . . . 3264
4.17.378 dorm2r . . . 3266
4.17.379 dorml2 . . . 3268
4.17.380 dormr2 . . . 3270
4.17.381 dormr3 . . . 3272
4.17.382 dpbtf2 . . . 3273
4.17.383 dpotf2 . . . 3275
4.17.384 dpstf2 . . . 3276
4.17.385 dptts2 . . . 3277
4.17.386 drscl . . . 3278
4.17.387 dsfrk . . . 3279
4.17.388 dsyconv . . . 3281
4.17.389 dsygs2 . . . 3282
4.17.390 dsyswapr . . . 3284
4.17.391 dsytd2 . . . 3285
4.17.392 dsytf2 . . . 3286
4.17.393 dsytf2_rook . . . 3287
4.17.394 dtfsm . . . 3289
4.17.395 dtfttp . . . 3291
4.17.396 dtfttr . . . 3292
4.17.397 dtgex2 . . . 3293
4.17.398 dtgsy2 . . . 3295
4.17.399 dtplqt2 . . . 3298
4.17.400 dtpqrt2 . . . 3300
4.17.401 dtprfb . . . 3301
4.17.402 dtpttf . . . 3303
4.17.403 dtpttr . . . 3304
4.17.404 dtrti2 . . . 3305
4.17.405 dtrttf . . . 3306
4.17.406 dtrttp . . . 3308
4.17.407 dzsum1 . . . 3309
4.17.408 scsum1 . . . 3309
4.17.409 sgbtf2 . . . 3310
4.17.410 sgebd2 . . . 3311
4.17.411 sgehd2 . . . 3313
4.17.412 sgelq2 . . . 3314
4.17.413 sgelqt3 . . . 3315
4.17.414 sgeql2 . . . 3316
4.17.415 sgeqr2 . . . 3317
4.17.416 sgeqr2p . . . 3319
4.17.417 sgeqrt2 . . . 3320
4.17.418 sgeqrt3 . . . 3321
4.17.419 sgerq2 . . . 3322

4.17.420 sgesc2 . . . 3323
4.17.421 sgetc2 . . . 3324
4.17.422 sgetf2 . . . 3325
4.17.423 sgsvj0 . . . 3326
4.17.424 sgsvj1 . . . 3329
4.17.425 sgtts2 . . . 3331
4.17.426 sisnan . . . 3333
4.17.427 sla_gbamv . . . 3333
4.17.428 sla_gbrcond . . . 3335
4.17.429 sla_gbrfsx_extended . . . 3337
4.17.430 sla_gbrpvgrw . . . 3342
4.17.431 sla_geamv . . . 3343
4.17.432 sla_gercond . . . 3345
4.17.433 sla_gerfsx_extended . . . 3347
4.17.434 sla_gerpvgrw . . . 3351
4.17.435 sla_lin_berr . . . 3352
4.17.436 sla_porcond . . . 3353
4.17.437 sla_porfsx_extended . . . 3355
4.17.438 sla_porpvgrw . . . 3359
4.17.439 sla_syamv . . . 3360
4.17.440 sla_syrcond . . . 3362
4.17.441 sla_syrfsx_extended . . . 3363
4.17.442 sla_syrpvgrw . . . 3368
4.17.443 sla_wwaddw . . . 3369
4.17.444 slabrd . . . 3370
4.17.445 slacn2 . . . 3371
4.17.446 slacon . . . 3373
4.17.447 slacpy . . . 3374
4.17.448 sladiv . . . 3375
4.17.449 slae2 . . . 3376
4.17.450 slaebz . . . 3377
4.17.451 slaed0 . . . 3380
4.17.452 slaed1 . . . 3382
4.17.453 slaed2 . . . 3383
4.17.454 slaed3 . . . 3386
4.17.455 slaed4 . . . 3388
4.17.456 slaed5 . . . 3389
4.17.457 slaed6 . . . 3390
4.17.458 slaed7 . . . 3391
4.17.459 slaed8 . . . 3394
4.17.460 slaed9 . . . 3397
4.17.461 slaeda . . . 3399
4.17.462 slaein . . . 3401
4.17.463 slaev2 . . . 3403
4.17.464 slaexc . . . 3404
4.17.465 slag2 . . . 3405
4.17.466 slag2d . . . 3407
4.17.467 slags2 . . . 3408
4.17.468 slagtf . . . 3410
4.17.469 slagtm . . . 3412
4.17.470 slagts . . . 3413
4.17.471 slagv2 . . . 3415
4.17.472 slahqr . . . 3417
4.17.473 slahr2 . . . 3419
4.17.474 slaic1 . . . 3420
4.17.475 slaisnan . . . 3422
4.17.476 slaln2 . . . 3423
4.17.477 slals0 . . . 3425

4.17.478 slalsa 3428
4.17.479 slalsd 3431
4.17.480 slamrg 3433
4.17.481 slamswlq 3434
4.17.482 slamtsqr 3436
4.17.483 slaneg 3438
4.17.484 slangb 3439
4.17.485 slange 3441
4.17.486 slangt 3442
4.17.487 slanhs 3443
4.17.488 slansb 3444
4.17.489 slansf 3445
4.17.490 slansp 3447
4.17.491 slanst 3448
4.17.492 slansy 3449
4.17.493 slantb 3450
4.17.494 slantp 3451
4.17.495 slantr 3453
4.17.496 slanv2 3454
4.17.497 slapll 3455
4.17.498 slapmr 3456
4.17.499 slapmt 3457
4.17.500 slapy2 3459
4.17.501 slapy3 3459
4.17.502 slaqgb 3460
4.17.503 slaqge 3461
4.17.504 slaqp2 3463
4.17.505 slaqps 3464
4.17.506 slaqr0 3466
4.17.507 slaqr1 3469
4.17.508 slaqr2 3470
4.17.509 slaqr3 3473
4.17.510 slaqr4 3476
4.17.511 slaqr5 3479
4.17.512 slaqsb 3482
4.17.513 slaqsp 3483
4.17.514 slaqsy 3484
4.17.515 slaqtr 3486
4.17.516 slar1v 3488
4.17.517 slar2v 3490
4.17.518 slarf 3491
4.17.519 slarfb 3493
4.17.520 slarfg 3495
4.17.521 slarfgp 3496
4.17.522 slarft 3497
4.17.523 slarfx 3498
4.17.524 slarfy 3500
4.17.525 slargv 3501
4.17.526 slarnv 3502
4.17.527 slarra 3503
4.17.528 slarrb 3504
4.17.529 slarrc 3507
4.17.530 slarrd 3508
4.17.531 slarre 3511
4.17.532 slarrf 3514
4.17.533 slarrj 3516
4.17.534 slarrk 3518
4.17.535 slarrr 3520

4.17.536 slarrv 3520
4.17.537 slarscl2 3524
4.17.538 slartg 3525
4.17.539 slartgp 3526
4.17.540 slartgs 3527
4.17.541 slartv 3528
4.17.542 slaruv 3529
4.17.543 slarz 3530
4.17.544 slarzb 3531
4.17.545 slarzt 3533
4.17.546 slas2 3535
4.17.547 slascl 3536
4.17.548 slascl2 3537
4.17.549 slasd0 3538
4.17.550 slasd1 3540
4.17.551 slasd2 3542
4.17.552 slasd3 3545
4.17.553 slasd4 3548
4.17.554 slasd5 3549
4.17.555 slasd6 3551
4.17.556 slasd7 3554
4.17.557 slasd8 3557
4.17.558 slasda 3559
4.17.559 slasdq 3562
4.17.560 slasdt 3564
4.17.561 slaset 3565
4.17.562 slasq1 3567
4.17.563 slasq2 3568
4.17.564 slasq3 3569
4.17.565 slasq4 3571
4.17.566 slasq5 3573
4.17.567 slasq6 3574
4.17.568 slasr 3576
4.17.569 slasrt 3578
4.17.570 slassq 3579
4.17.571 slasv2 3580
4.17.572 slaswlq 3581
4.17.573 slaswp 3583
4.17.574 slasy2 3584
4.17.575 slasyf 3586
4.17.576 slasyf_aa 3588
4.17.577 slasyf_rook 3589
4.17.578 slatbs 3591
4.17.579 slatdf 3593
4.17.580 slatps 3594
4.17.581 slatrd 3596
4.17.582 slatrs 3597
4.17.583 slatrz 3599
4.17.584 slatsqr 3600
4.17.585 slauu2 3602
4.17.586 slauum 3603
4.17.587 sorbdb1 3604
4.17.588 sorbdb2 3606
4.17.589 sorbdb3 3608
4.17.590 sorbdb4 3610
4.17.591 sorbdb5 3612
4.17.592 sorbdb6 3614
4.17.593 sorg2l 3616

4.17.594 sorg2r 3617
4.17.595 sorgl2 3619
4.17.596 sorgr2 3620
4.17.597 sorm2l 3621
4.17.598 sorm2r 3623
4.17.599 sorml2 3625
4.17.600 sormr2 3627
4.17.601 sormr3 3628
4.17.602 spbtf2 3630
4.17.603 spotf2 3631
4.17.604 spstf2 3633
4.17.605 sptts2 3634
4.17.606 srscl 3635
4.17.607 ssfrk 3636
4.17.608 ssyconv 3638
4.17.609 ssygs2 3639
4.17.610 ssyswapr 3640
4.17.611 ssytd2 3641
4.17.612 ssytf2 3643
4.17.613 ssytf2_rook 3644
4.17.614 stfsm 3646
4.17.615 stfttp 3648
4.17.616 stfttr 3649
4.17.617 stgex2 3650
4.17.618 stgsy2 3652
4.17.619 stplqt2 3655
4.17.620 stpqrt2 3656
4.17.621 stprfb 3658
4.17.622 stpttf 3660
4.17.623 stpttr 3661
4.17.624 strti2 3662
4.17.625 strttf 3663
4.17.626 strttp 3664
4.17.627 zdrscl 3665
4.17.628 zgbtf2 3666
4.17.629 zgebd2 3668
4.17.630 zgehd2 3669
4.17.631 zgelq2 3670
4.17.632 zgelqt3 3672
4.17.633 zgeql2 3673
4.17.634 zgeqr2 3674
4.17.635 zgeqr2p 3675
4.17.636 zgeqrt2 3676
4.17.637 zgeqrt3 3677
4.17.638 zgerq2 3679
4.17.639 zgesc2 3680
4.17.640 zgetc2 3681
4.17.641 zgetf2 3682
4.17.642 zgsvj0 3683
4.17.643 zgsvj1 3685
4.17.644 zgtts2 3688
4.17.645 zhegs2 3689
4.17.646 zheswapr 3691
4.17.647 zhetd2 3692
4.17.648 zhetf2 3693
4.17.649 zhetf2_rook 3695
4.17.650 zhfrk 3696
4.17.651 zla_gbamv 3698

4.17.652 zla_gbrcond_c . . . 3700
4.17.653 zla_gbrcond_x . . . 3702
4.17.654 zla_gbrfsx_extended . . . 3703
4.17.655 zla_gbrpvgrw . . . 3708
4.17.656 zla_geamv . . . 3709
4.17.657 zla_gercond_c . . . 3711
4.17.658 zla_gercond_x . . . 3713
4.17.659 zla_gerfsx_extended . . . 3714
4.17.660 zla_gerpvgrw . . . 3719
4.17.661 zla_heamv . . . 3720
4.17.662 zla_hercond_c . . . 3721
4.17.663 zla_hercond_x . . . 3723
4.17.664 zla_herfsx_extended . . . 3724
4.17.665 zla_herpvgrw . . . 3729
4.17.666 zla_lin_berr . . . 3730
4.17.667 zla_porcond_c . . . 3731
4.17.668 zla_porcond_x . . . 3733
4.17.669 zla_porfsx_extended . . . 3734
4.17.670 zla_porpvgrw . . . 3738
4.17.671 zla_syamv . . . 3740
4.17.672 zla_syrcond_c . . . 3741
4.17.673 zla_syrcond_x . . . 3743
4.17.674 zla_syrfsx_extended . . . 3744
4.17.675 zla_syrpvgrw . . . 3749
4.17.676 zla_wwaddw . . . 3750
4.17.677 zlabrd . . . 3751
4.17.678 zlacgv . . . 3753
4.17.679 zlacn2 . . . 3754
4.17.680 zlacon . . . 3755
4.17.681 zlacp2 . . . 3756
4.17.682 zlacpy . . . 3757
4.17.683 zlacrm . . . 3758
4.17.684 zlacrt . . . 3759
4.17.685 zladiv . . . 3760
4.17.686 zlaed0 . . . 3761
4.17.687 zlaed7 . . . 3762
4.17.688 zlaed8 . . . 3765
4.17.689 zlaein . . . 3768
4.17.690 zlaesy . . . 3770
4.17.691 zlaev2 . . . 3771
4.17.692 zlag2c . . . 3772
4.17.693 zlags2 . . . 3773
4.17.694 zlagtm . . . 3775
4.17.695 zlahef . . . 3777
4.17.696 zlahef_aa . . . 3779
4.17.697 zlahef_rook . . . 3780
4.17.698 zlahqr . . . 3782
4.17.699 zlahr2 . . . 3784
4.17.700 zlaic1 . . . 3786
4.17.701 zlals0 . . . 3787
4.17.702 zlalsa . . . 3791
4.17.703 zlalsd . . . 3794
4.17.704 zlamswlq . . . 3796
4.17.705 zlamtsqr . . . 3798
4.17.706 zlangb . . . 3800
4.17.707 zlange . . . 3801
4.17.708 zlangt . . . 3802
4.17.709 zlanhb . . . 3803

4.17.710 zlanhe 3805
4.17.711 zlanhf 3806
4.17.712 zlanhp 3807
4.17.713 zlanhs 3808
4.17.714 zlanht 3810
4.17.715 zlansb 3811
4.17.716 zlansp 3812
4.17.717 zlansy 3813
4.17.718 zlantb 3814
4.17.719 zlantp 3816
4.17.720 zlantr 3817
4.17.721 zlapll 3818
4.17.722 zlapmr 3820
4.17.723 zlapmt 3821
4.17.724 zlaqgb 3822
4.17.725 zlaqge 3823
4.17.726 zlaqhb 3825
4.17.727 zlaqhe 3826
4.17.728 zlaqhp 3827
4.17.729 zlaqp2 3829
4.17.730 zlaqps 3830
4.17.731 zlaqr0 3832
4.17.732 zlaqr1 3835
4.17.733 zlaqr2 3836
4.17.734 zlaqr3 3839
4.17.735 zlaqr4 3842
4.17.736 zlaqr5 3844
4.17.737 zlaqsb 3847
4.17.738 zlaqsp 3849
4.17.739 zlaqsy 3850
4.17.740 zlar1v 3851
4.17.741 zlar2v 3854
4.17.742 zlarcm 3855
4.17.743 zlarf 3856
4.17.744 zlarfb 3858
4.17.745 zlarfg 3859
4.17.746 zlarfgp 3861
4.17.747 zlarft 3862
4.17.748 zlarfx 3863
4.17.749 zlarfy 3865
4.17.750 zlargv 3866
4.17.751 zlarnv 3867
4.17.752 zlarrv 3868
4.17.753 zlarscl2 3871
4.17.754 zlartg 3872
4.17.755 zlartv 3873
4.17.756 zlarz 3874
4.17.757 zlarzb 3876
4.17.758 zlarzt 3878
4.17.759 zlascl 3879
4.17.760 zlascl2 3881
4.17.761 zlaset 3882
4.17.762 zlasr 3883
4.17.763 zlassq 3885
4.17.764 zlaswlq 3886
4.17.765 zlaswp 3888
4.17.766 zlasyf 3889
4.17.767 zlasyf_aa 3891

4.17.768 zlasyf_rook . . . 3893
4.17.769 zlat2c . . . 3895
4.17.770 zlatbs . . . 3896
4.17.771 zlatdf . . . 3898
4.17.772 zlatps . . . 3899
4.17.773 zlatrd . . . 3901
4.17.774 zlatrs . . . 3902
4.17.775 zlatrz . . . 3904
4.17.776 zlatsqr . . . 3905
4.17.777 zlauu2 . . . 3907
4.17.778 zlauum . . . 3908
4.17.779 zpbtf2 . . . 3909
4.17.780 zpotf2 . . . 3910
4.17.781 zpstf2 . . . 3911
4.17.782 zptts2 . . . 3913
4.17.783 zsyconv . . . 3914
4.17.784 zsyswapr . . . 3915
4.17.785 zsytf2 . . . 3916
4.17.786 zsytf2_rook . . . 3918
4.17.787 ztfsm . . . 3919
4.17.788 ztfttp . . . 3921
4.17.789 ztfttr . . . 3922
4.17.790 ztgex2 . . . 3923
4.17.791 ztgsy2 . . . 3925
4.17.792 ztplqt2 . . . 3928
4.17.793 ztpqrt2 . . . 3929
4.17.794 ztprfb . . . 3931
4.17.795 ztpttf . . . 3933
4.17.796 ztpttr . . . 3934
4.17.797 ztrti2 . . . 3935
4.17.798 ztrttf . . . 3936
4.17.799 ztrttp . . . 3938
4.17.800 zunbdb1 . . . 3939
4.17.801 zunbdb2 . . . 3941
4.17.802 zunbdb3 . . . 3943
4.17.803 zunbdb4 . . . 3945
4.17.804 zunbdb5 . . . 3947
4.17.805 zunbdb6 . . . 3949
4.17.806 zung2l . . . 3951
4.17.807 zung2r . . . 3952
4.17.808 zungl2 . . . 3954
4.17.809 zungr2 . . . 3955
4.17.810 zunm2l . . . 3956
4.17.811 zunm2r . . . 3958
4.17.812 zunml2 . . . 3960
4.17.813 zunmr2 . . . 3962
4.17.814 zunmr3 . . . 3964
4.18 LAPACK utilities routines . . . 3966
4.18.1 chla_transtype . . . 3966
4.18.2 dlabad . . . 3966
4.18.3 ieeeck . . . 3967
4.18.4 iladiag . . . 3967
4.18.5 ilaenv . . . 3968
4.18.6 ilaprec . . . 3969
4.18.7 ilatrans . . . 3970
4.18.8 ilauplo . . . 3970
4.18.9 iparmq . . . 3970
4.18.10 slabad . . . 3972

4.18.11 xerbla . . . 3973
4.18.12 xerbla_array . . . 3973
4.19 LAPACKE - C interfaces for LAPACK functions . . . 3974
4.19.1 LAPACKE_cbbcsd . . . 3974
4.19.2 LAPACKE_cbbcsd_work . . . 3975
4.19.3 LAPACKE_cbdsqr . . . 3975
4.19.4 LAPACKE_cbdsqr_work . . . 3976
4.19.5 LAPACKE_cgbbrd . . . 3977
4.19.6 LAPACKE_cgbbrd_work . . . 3977
4.19.7 LAPACKE_cgbcon . . . 3978
4.19.8 LAPACKE_cgbcon_work . . . 3978
4.19.9 LAPACKE_cgbequ . . . 3979
4.19.10 LAPACKE_cgbequ_work . . . 3979
4.19.11 LAPACKE_cgbequb . . . 3980
4.19.12 LAPACKE_cgbequb_work . . . 3980
4.19.13 LAPACKE_cgbrfs . . . 3981
4.19.14 LAPACKE_cgbrfs_work . . . 3982
4.19.15 LAPACKE_cgbrfsx . . . 3982
4.19.16 LAPACKE_cgbrfsx_work . . . 3983
4.19.17 LAPACKE_cgbsv . . . 3983
4.19.18 LAPACKE_cgbsv_work . . . 3984
4.19.19 LAPACKE_cgbsvx . . . 3985
4.19.20 LAPACKE_cgbsvx_work . . . 3985
4.19.21 LAPACKE_cgbsvxx . . . 3986
4.19.22 LAPACKE_cgbsvxx_work . . . 3986
4.19.23 LAPACKE_cgbtrf . . . 3987
4.19.24 LAPACKE_cgbtrf_work . . . 3988
4.19.25 LAPACKE_cgbtrs . . . 3988
4.19.26 LAPACKE_cgbtrs_work . . . 3989
4.19.27 LAPACKE_cgebak . . . 3989
4.19.28 LAPACKE_cgebak_work . . . 3990
4.19.29 LAPACKE_cgebal . . . 3990
4.19.30 LAPACKE_cgebal_work . . . 3991
4.19.31 LAPACKE_cgebrd . . . 3991
4.19.32 LAPACKE_cgebrd_work . . . 3992
4.19.33 LAPACKE_cgecon . . . 3992
4.19.34 LAPACKE_cgecon_work . . . 3993
4.19.35 LAPACKE_cgeequ . . . 3993
4.19.36 LAPACKE_cgeequ_work . . . 3994
4.19.37 LAPACKE_cgeequb . . . 3994
4.19.38 LAPACKE_cgeequb_work . . . 3995
4.19.39 LAPACKE_cgees . . . 3995
4.19.40 LAPACKE_cgees_work . . . 3996
4.19.41 LAPACKE_cgeesx . . . 3996
4.19.42 LAPACKE_cgeesx_work . . . 3997
4.19.43 LAPACKE_cgeev . . . 3997
4.19.44 LAPACKE_cgeev_work . . . 3998
4.19.45 LAPACKE_cgeevx . . . 3998
4.19.46 LAPACKE_cgeevx_work . . . 3999
4.19.47 LAPACKE_cgehrd . . . 4000
4.19.48 LAPACKE_cgehrd_work . . . 4000
4.19.49 LAPACKE_cgejsv . . . 4001
4.19.50 LAPACKE_cgejsv_work . . . 4001
4.19.51 LAPACKE_cgelq . . . 4002
4.19.52 LAPACKE_cgelq2 . . . 4002
4.19.53 LAPACKE_cgelq2_work . . . 4003
4.19.54 LAPACKE_cgelq_work . . . 4003
4.19.55 LAPACKE_cgelqf . . . 4004

4.19.56 LAPACKE_cgelqf_work 4004
4.19.57 LAPACKE_cgels 4005
4.19.58 LAPACKE_cgels_work 4005
4.19.59 LAPACKE_cgelsd 4006
4.19.60 LAPACKE_cgelsd_work 4006
4.19.61 LAPACKE_cgelss 4007
4.19.62 LAPACKE_cgelss_work 4008
4.19.63 LAPACKE_cgelsy 4008
4.19.64 LAPACKE_cgelsy_work 4009
4.19.65 LAPACKE_cgemlq 4009
4.19.66 LAPACKE_cgemlq_work 4010
4.19.67 LAPACKE_cgemqr 4010
4.19.68 LAPACKE_cgemqr_work 4011
4.19.69 LAPACKE_cgemqrt 4011
4.19.70 LAPACKE_cgemqrt_work 4012
4.19.71 LAPACKE_cgeqlf 4013
4.19.72 LAPACKE_cgeqlf_work 4013
4.19.73 LAPACKE_cgeqp3 4014
4.19.74 LAPACKE_cgeqp3_work 4014
4.19.75 LAPACKE_cgeqpf 4015
4.19.76 LAPACKE_cgeqpf_work 4015
4.19.77 LAPACKE_cgeqr 4016
4.19.78 LAPACKE_cgeqr2 4016
4.19.79 LAPACKE_cgeqr2_work 4017
4.19.80 LAPACKE_cgeqr_work 4017
4.19.81 LAPACKE_cgeqrf 4018
4.19.82 LAPACKE_cgeqrf_work 4018
4.19.83 LAPACKE_cgeqrfp 4019
4.19.84 LAPACKE_cgeqrfp_work 4019
4.19.85 LAPACKE_cgeqrt 4020
4.19.86 LAPACKE_cgeqrt2 4020
4.19.87 LAPACKE_cgeqrt2_work 4021
4.19.88 LAPACKE_cgeqrt3 4021
4.19.89 LAPACKE_cgeqrt3_work 4022
4.19.90 LAPACKE_cgeqrt_work 4022
4.19.91 LAPACKE_cgerfs 4023
4.19.92 LAPACKE_cgerfs_work 4023
4.19.93 LAPACKE_cgerfsx 4024
4.19.94 LAPACKE_cgerfsx_work 4024
4.19.95 LAPACKE_cgerqf 4025
4.19.96 LAPACKE_cgerqf_work 4026
4.19.97 LAPACKE_cgesdd 4026
4.19.98 LAPACKE_cgesdd_work 4027
4.19.99 LAPACKE_cgesv 4027
4.19.100 LAPACKE_cgesv_work 4028
4.19.101 LAPACKE_cgesvd 4028
4.19.102 LAPACKE_cgesvd_work 4029
4.19.103 LAPACKE_cgesvdx 4029
4.19.104 LAPACKE_cgesvdx_work 4030
4.19.105 LAPACKE_cgesvj 4031
4.19.106 LAPACKE_cgesvj_work 4031
4.19.107 LAPACKE_cgesvx 4032
4.19.108 LAPACKE_cgesvx_work 4032
4.19.109 LAPACKE_cgesvxx 4033
4.19.110 LAPACKE_cgesvxx_work 4034
4.19.111 LAPACKE_cgetf2 4034
4.19.112 LAPACKE_cgetf2_work 4035
4.19.113 LAPACKE_cgetrf 4035

4.19.114 LAPACKE_cgetrf2 4036
4.19.115 LAPACKE_cgetrf2_work 4036
4.19.116 LAPACKE_cgetrf_work 4037
4.19.117 LAPACKE_cgetri 4037
4.19.118 LAPACKE_cgetri_work 4038
4.19.119 LAPACKE_cgetrs 4038
4.19.120 LAPACKE_cgetrs_work 4039
4.19.121 LAPACKE_cgetsls 4039
4.19.122 LAPACKE_cgetsls_work 4040
4.19.123 LAPACKE_cggbak 4040
4.19.124 LAPACKE_cggbak_work 4041
4.19.125 LAPACKE_cggbal 4041
4.19.126 LAPACKE_cggbal_work 4042
4.19.127 LAPACKE_cgges 4042
4.19.128 LAPACKE_cgges3 4043
4.19.129 LAPACKE_cgges3_work 4043
4.19.130 LAPACKE_cgges_work 4044
4.19.131 LAPACKE_cggesx 4045
4.19.132 LAPACKE_cggesx_work 4045
4.19.133 LAPACKE_cggev 4046
4.19.134 LAPACKE_cggev3 4047
4.19.135 LAPACKE_cggev3_work 4047
4.19.136 LAPACKE_cggev_work 4048
4.19.137 LAPACKE_cggevx 4048
4.19.138 LAPACKE_cggevx_work 4049
4.19.139 LAPACKE_cggglm 4050
4.19.140 LAPACKE_cggglm_work 4050
4.19.141 LAPACKE_cgghd3 4051
4.19.142 LAPACKE_cgghd3_work 4051
4.19.143 LAPACKE_cgghrd 4052
4.19.144 LAPACKE_cgghrd_work 4053
4.19.145 LAPACKE_cgglse 4053
4.19.146 LAPACKE_cgglse_work 4054
4.19.147 LAPACKE_cggqrf 4054
4.19.148 LAPACKE_cggqrf_work 4055
4.19.149 LAPACKE_cggrqf 4055
4.19.150 LAPACKE_cggrqf_work 4056
4.19.151 LAPACKE_cggsvd 4056
4.19.152 LAPACKE_cggsvd3 4057
4.19.153 LAPACKE_cggsvd3_work 4058
4.19.154 LAPACKE_cggsvd_work 4058
4.19.155 LAPACKE_cggsvp 4059
4.19.156 LAPACKE_cggsvp3 4060
4.19.157 LAPACKE_cggsvp3_work 4060
4.19.158 LAPACKE_cggsvp_work 4061
4.19.159 LAPACKE_cgtcon 4061
4.19.160 LAPACKE_cgtcon_work 4062
4.19.161 LAPACKE_cgtrfs 4063
4.19.162 LAPACKE_cgtrfs_work 4063
4.19.163 LAPACKE_cgtsv 4064
4.19.164 LAPACKE_cgtsv_work 4064
4.19.165 LAPACKE_cgtsvx 4065
4.19.166 LAPACKE_cgtsvx_work 4065
4.19.167 LAPACKE_cgttrf 4066
4.19.168 LAPACKE_cgttrf_work 4067
4.19.169 LAPACKE_cgttrs 4067
4.19.170 LAPACKE_cgttrs_work 4068
4.19.171 LAPACKE_chbev 4068

4.19.172 LAPACKE_chbev_2stage 4069
4.19.173 LAPACKE_chbev_2stage_work 4069
4.19.174 LAPACKE_chbev_work 4070
4.19.175 LAPACKE_chbevd 4070
4.19.176 LAPACKE_chbevd_2stage 4071
4.19.177 LAPACKE_chbevd_2stage_work 4071
4.19.178 LAPACKE_chbevd_work 4072
4.19.179 LAPACKE_chbevx 4073
4.19.180 LAPACKE_chbevx_2stage 4073
4.19.181 LAPACKE_chbevx_2stage_work 4074
4.19.182 LAPACKE_chbevx_work 4074
4.19.183 LAPACKE_chbgst 4075
4.19.184 LAPACKE_chbgst_work 4076
4.19.185 LAPACKE_chbgv 4076
4.19.186 LAPACKE_chbgv_work 4077
4.19.187 LAPACKE_chbgvd 4077
4.19.188 LAPACKE_chbgvd_work 4078
4.19.189 LAPACKE_chbgvx 4078
4.19.190 LAPACKE_chbgvx_work 4079
4.19.191 LAPACKE_chbtrd 4080
4.19.192 LAPACKE_chbtrd_work 4080
4.19.193 LAPACKE_checon 4081
4.19.194 LAPACKE_checon_3 4081
4.19.195 LAPACKE_checon_3_work 4082
4.19.196 LAPACKE_checon_work 4082
4.19.197 LAPACKE_cheequb 4083
4.19.198 LAPACKE_cheequb_work 4083
4.19.199 LAPACKE_cheev 4084
4.19.200 LAPACKE_cheev_2stage 4084
4.19.201 LAPACKE_cheev_2stage_work 4085
4.19.202 LAPACKE_cheev_work 4085
4.19.203 LAPACKE_cheevd 4086
4.19.204 LAPACKE_cheevd_2stage 4086
4.19.205 LAPACKE_cheevd_2stage_work 4087
4.19.206 LAPACKE_cheevd_work 4087
4.19.207 LAPACKE_cheevr 4088
4.19.208 LAPACKE_cheevr_2stage 4088
4.19.209 LAPACKE_cheevr_2stage_work 4089
4.19.210 LAPACKE_cheevr_work 4089
4.19.211 LAPACKE_cheevx 4090
4.19.212 LAPACKE_cheevx_2stage 4091
4.19.213 LAPACKE_cheevx_2stage_work 4091
4.19.214 LAPACKE_cheevx_work 4092
4.19.215 LAPACKE_chegst 4092
4.19.216 LAPACKE_chegst_work 4093
4.19.217 LAPACKE_chegv 4093
4.19.218 LAPACKE_chegv_2stage 4094
4.19.219 LAPACKE_chegv_2stage_work 4094
4.19.220 LAPACKE_chegv_work 4095
4.19.221 LAPACKE_chegvd 4096
4.19.222 LAPACKE_chegvd_work 4096
4.19.223 LAPACKE_chegvx 4097
4.19.224 LAPACKE_chegvx_work 4097
4.19.225 LAPACKE_cherfs 4098
4.19.226 LAPACKE_cherfs_work 4098
4.19.227 LAPACKE_cherfsx 4099
4.19.228 LAPACKE_cherfsx_work 4100
4.19.229 LAPACKE_chesv 4100

4.19.230 LAPACKE_chesv_aa 4101
4.19.231 LAPACKE_chesv_aa_2stage 4101
4.19.232 LAPACKE_chesv_aa_work 4102
4.19.233 LAPACKE_chesv_rk 4103
4.19.234 LAPACKE_chesv_rk_work 4103
4.19.235 LAPACKE_chesv_work 4104
4.19.236 LAPACKE_chesvx 4104
4.19.237 LAPACKE_chesvx_work 4105
4.19.238 LAPACKE_chesvxx 4105
4.19.239 LAPACKE_chesvxx_work 4106
4.19.240 LAPACKE_cheswapr 4107
4.19.241 LAPACKE_cheswapr_work 4107
4.19.242 LAPACKE_chetrd 4108
4.19.243 LAPACKE_chetrd_work 4108
4.19.244 LAPACKE_chetrf 4109
4.19.245 LAPACKE_chetrf_aa 4109
4.19.246 LAPACKE_chetrf_aa_2stage 4110
4.19.247 LAPACKE_chetrf_aa_work 4110
4.19.248 LAPACKE_chetrf_rk 4111
4.19.249 LAPACKE_chetrf_rk_work 4111
4.19.250 LAPACKE_chetrf_rook 4112
4.19.251 LAPACKE_chetrf_rook_work 4112
4.19.252 LAPACKE_chetrf_work 4113
4.19.253 LAPACKE_chetri 4113
4.19.254 LAPACKE_chetri2 4114
4.19.255 LAPACKE_chetri2_work 4114
4.19.256 LAPACKE_chetri2x 4115
4.19.257 LAPACKE_chetri2x_work 4115
4.19.258 LAPACKE_chetri_3 4116
4.19.259 LAPACKE_chetri_3_work 4116
4.19.260 LAPACKE_chetri_work 4117
4.19.261 LAPACKE_chetrs 4117
4.19.262 LAPACKE_chetrs2 4118
4.19.263 LAPACKE_chetrs2_work 4118
4.19.264 LAPACKE_chetrs_3 4119
4.19.265 LAPACKE_chetrs_3_work 4119
4.19.266 LAPACKE_chetrs_aa 4120
4.19.267 LAPACKE_chetrs_aa_2stage 4120
4.19.268 LAPACKE_chetrs_aa_work 4121
4.19.269 LAPACKE_chetrs_rook 4121
4.19.270 LAPACKE_chetrs_rook_work 4122
4.19.271 LAPACKE_chetrs_work 4122
4.19.272 LAPACKE_chfrk 4123
4.19.273 LAPACKE_chfrk_work 4123
4.19.274 LAPACKE_chgeqz 4124
4.19.275 LAPACKE_chgeqz_work 4124
4.19.276 LAPACKE_chpcon 4125
4.19.277 LAPACKE_chpcon_work 4126
4.19.278 LAPACKE_chpev 4126
4.19.279 LAPACKE_chpev_work 4127
4.19.280 LAPACKE_chpevd 4127
4.19.281 LAPACKE_chpevd_work 4128
4.19.282 LAPACKE_chpevx 4128
4.19.283 LAPACKE_chpevx_work 4129
4.19.284 LAPACKE_chpgst 4129
4.19.285 LAPACKE_chpgst_work 4130
4.19.286 LAPACKE_chpgv 4130
4.19.287 LAPACKE_chpgv_work 4131

4.19.288 LAPACKE_chpgvd 4131
4.19.289 LAPACKE_chpgvd_work 4132
4.19.290 LAPACKE_chpgvx 4132
4.19.291 LAPACKE_chpgvx_work 4133
4.19.292 LAPACKE_chprfs 4133
4.19.293 LAPACKE_chprfs_work 4134
4.19.294 LAPACKE_chpsv 4135
4.19.295 LAPACKE_chpsv_work 4135
4.19.296 LAPACKE_chpsvx 4136
4.19.297 LAPACKE_chpsvx_work 4136
4.19.298 LAPACKE_chptrd 4137
4.19.299 LAPACKE_chptrd_work 4137
4.19.300 LAPACKE_chptrf 4138
4.19.301 LAPACKE_chptrf_work 4138
4.19.302 LAPACKE_chptri 4139
4.19.303 LAPACKE_chptri_work 4139
4.19.304 LAPACKE_chptrs 4140
4.19.305 LAPACKE_chptrs_work 4140
4.19.306 LAPACKE_chsein 4141
4.19.307 LAPACKE_chsein_work 4141
4.19.308 LAPACKE_chseqr 4142
4.19.309 LAPACKE_chseqr_work 4142
4.19.310 LAPACKE_clacgv 4143
4.19.311 LAPACKE_clacgv_work 4143
4.19.312 LAPACKE_clacn2 4144
4.19.313 LAPACKE_clacn2_work 4144
4.19.314 LAPACKE_clacp2 4145
4.19.315 LAPACKE_clacp2_work 4145
4.19.316 LAPACKE_clacpy 4146
4.19.317 LAPACKE_clacpy_work 4146
4.19.318 LAPACKE_clacrm 4147
4.19.319 LAPACKE_clacrm_work 4147
4.19.320 LAPACKE_clag2z 4148
4.19.321 LAPACKE_clag2z_work 4148
4.19.322 LAPACKE_clagge 4149
4.19.323 LAPACKE_clagge_work 4149
4.19.324 LAPACKE_claghe 4149
4.19.325 LAPACKE_claghe_work 4149
4.19.326 LAPACKE_clagsy 4149
4.19.327 LAPACKE_clagsy_work 4149
4.19.328 LAPACKE_clange 4150
4.19.329 LAPACKE_clange_work 4150
4.19.330 LAPACKE_clanhe 4151
4.19.331 LAPACKE_clanhe_work 4151
4.19.332 LAPACKE_clansy 4152
4.19.333 LAPACKE_clansy_work 4152
4.19.334 LAPACKE_clantr 4153
4.19.335 LAPACKE_clantr_work 4153
4.19.336 LAPACKE_clapmr 4154
4.19.337 LAPACKE_clapmr_work 4154
4.19.338 LAPACKE_clapmt 4155
4.19.339 LAPACKE_clapmt_work 4155
4.19.340 LAPACKE_clarcm 4155
4.19.341 LAPACKE_clarcm_work 4156
4.19.342 LAPACKE_clarfb 4156
4.19.343 LAPACKE_clarfb_work 4157
4.19.344 LAPACKE_clarfg 4157
4.19.345 LAPACKE_clarfg_work 4158

4.19.346 LAPACKE_clarft 4158
4.19.347 LAPACKE_clarft_work 4159
4.19.348 LAPACKE_clarfx 4159
4.19.349 LAPACKE_clarfx_work 4160
4.19.350 LAPACKE_clarnv 4160
4.19.351 LAPACKE_clarnv_work 4161
4.19.352 LAPACKE_clascl 4161
4.19.353 LAPACKE_clascl_work 4162
4.19.354 LAPACKE_claset 4162
4.19.355 LAPACKE_claset_work 4163
4.19.356 LAPACKE_classq 4163
4.19.357 LAPACKE_classq_work 4164
4.19.358 LAPACKE_claswp 4164
4.19.359 LAPACKE_claswp_work 4165
4.19.360 LAPACKE_clatms 4165
4.19.361 LAPACKE_clatms_work 4165
4.19.362 LAPACKE_clauum 4166
4.19.363 LAPACKE_clauum_work 4166
4.19.364 LAPACKE_cpbcon 4167
4.19.365 LAPACKE_cpbcon_work 4167
4.19.366 LAPACKE_cpbequ 4168
4.19.367 LAPACKE_cpbequ_work 4168
4.19.368 LAPACKE_cpbrfs 4169
4.19.369 LAPACKE_cpbrfs_work 4169
4.19.370 LAPACKE_cpbstf 4170
4.19.371 LAPACKE_cpbstf_work 4170
4.19.372 LAPACKE_cpbsv 4171
4.19.373 LAPACKE_cpbsv_work 4171
4.19.374 LAPACKE_cpbsvx 4172
4.19.375 LAPACKE_cpbsvx_work 4172
4.19.376 LAPACKE_cpbtrf 4173
4.19.377 LAPACKE_cpbtrf_work 4173
4.19.378 LAPACKE_cpbtrs 4174
4.19.379 LAPACKE_cpbtrs_work 4175
4.19.380 LAPACKE_cpftrf 4175
4.19.381 LAPACKE_cpftrf_work 4176
4.19.382 LAPACKE_cpftri 4176
4.19.383 LAPACKE_cpftri_work 4177
4.19.384 LAPACKE_cpftrs 4177
4.19.385 LAPACKE_cpftrs_work 4178
4.19.386 LAPACKE_cpocon 4178
4.19.387 LAPACKE_cpocon_work 4178
4.19.388 LAPACKE_cpoequ 4179
4.19.389 LAPACKE_cpoequ_work 4180
4.19.390 LAPACKE_cpoequb 4180
4.19.391 LAPACKE_cpoequb_work 4181
4.19.392 LAPACKE_cporfs 4181
4.19.393 LAPACKE_cporfs_work 4182
4.19.394 LAPACKE_cporfsx 4182
4.19.395 LAPACKE_cporfsx_work 4183
4.19.396 LAPACKE_cposv 4184
4.19.397 LAPACKE_cposv_work 4184
4.19.398 LAPACKE_cposvx 4185
4.19.399 LAPACKE_cposvx_work 4185
4.19.400 LAPACKE_cposvxx 4186
4.19.401 LAPACKE_cposvxx_work 4186
4.19.402 LAPACKE_cpotrf 4187
4.19.403 LAPACKE_cpotrf2 4188

4.19.404 LAPACKE_cpotrf2_work 4188
4.19.405 LAPACKE_cpotrf_work 4189
4.19.406 LAPACKE_cpotri 4189
4.19.407 LAPACKE_cpotri_work 4190
4.19.408 LAPACKE_cpotrs 4190
4.19.409 LAPACKE_cpotrs_work 4191
4.19.410 LAPACKE_cppcon 4191
4.19.411 LAPACKE_cppcon_work 4192
4.19.412 LAPACKE_cppequ 4192
4.19.413 LAPACKE_cppequ_work 4193
4.19.414 LAPACKE_cpprfs 4193
4.19.415 LAPACKE_cpprfs_work 4194
4.19.416 LAPACKE_cppsv 4194
4.19.417 LAPACKE_cppsv_work 4195
4.19.418 LAPACKE_cppsvx 4195
4.19.419 LAPACKE_cppsvx_work 4196
4.19.420 LAPACKE_cpptrf 4196
4.19.421 LAPACKE_cpptrf_work 4197
4.19.422 LAPACKE_cpptri 4197
4.19.423 LAPACKE_cpptri_work 4198
4.19.424 LAPACKE_cpptrs 4198
4.19.425 LAPACKE_cpptrs_work 4199
4.19.426 LAPACKE_cpstrf 4200
4.19.427 LAPACKE_cpstrf_work 4200
4.19.428 LAPACKE_cptcon 4201
4.19.429 LAPACKE_cptcon_work 4201
4.19.430 LAPACKE_cpteqr 4202
4.19.431 LAPACKE_cpteqr_work 4202
4.19.432 LAPACKE_cptrfs 4203
4.19.433 LAPACKE_cptrfs_work 4203
4.19.434 LAPACKE_cptsv 4204
4.19.435 LAPACKE_cptsv_work 4204
4.19.436 LAPACKE_cptsvx 4205
4.19.437 LAPACKE_cptsvx_work 4205
4.19.438 LAPACKE_cpttrf 4206
4.19.439 LAPACKE_cpttrf_work 4206
4.19.440 LAPACKE_cpttrs 4207
4.19.441 LAPACKE_cpttrs_work 4207
4.19.442 LAPACKE_cspcon 4208
4.19.443 LAPACKE_cspcon_work 4208
4.19.444 LAPACKE_csprfs 4209
4.19.445 LAPACKE_csprfs_work 4209
4.19.446 LAPACKE_cspsv 4210
4.19.447 LAPACKE_cspsv_work 4211
4.19.448 LAPACKE_cspsvx 4211
4.19.449 LAPACKE_cspsvx_work 4212
4.19.450 LAPACKE_csptrf 4212
4.19.451 LAPACKE_csptrf_work 4213
4.19.452 LAPACKE_csptri 4213
4.19.453 LAPACKE_csptri_work 4214
4.19.454 LAPACKE_csptrs 4214
4.19.455 LAPACKE_csptrs_work 4215
4.19.456 LAPACKE_cstedc 4215
4.19.457 LAPACKE_cstedc_work 4216
4.19.458 LAPACKE_cstegr 4216
4.19.459 LAPACKE_cstegr_work 4217
4.19.460 LAPACKE_cstein 4217
4.19.461 LAPACKE_cstein_work 4218

4.19.462 LAPACKE_cstemr 4219
4.19.463 LAPACKE_cstemr_work 4219
4.19.464 LAPACKE_csteqr 4220
4.19.465 LAPACKE_csteqr_work 4220
4.19.466 LAPACKE_csycon 4221
4.19.467 LAPACKE_csycon_3 4221
4.19.468 LAPACKE_csycon_3_work 4222
4.19.469 LAPACKE_csycon_work 4222
4.19.470 LAPACKE_csyconv 4223
4.19.471 LAPACKE_csyconv_work 4223
4.19.472 LAPACKE_csyequb 4224
4.19.473 LAPACKE_csyequb_work 4224
4.19.474 LAPACKE_csyr 4225
4.19.475 LAPACKE_csyr_work 4225
4.19.476 LAPACKE_csyrfs 4226
4.19.477 LAPACKE_csyrfs_work 4226
4.19.478 LAPACKE_csyrfsx 4227
4.19.479 LAPACKE_csyrfsx_work 4228
4.19.480 LAPACKE_csysv 4228
4.19.481 LAPACKE_csysv_aa 4229
4.19.482 LAPACKE_csysv_aa_2stage 4229
4.19.483 LAPACKE_csysv_aa_work 4230
4.19.484 LAPACKE_csysv_rk 4231
4.19.485 LAPACKE_csysv_rk_work 4231
4.19.486 LAPACKE_csysv_rook 4232
4.19.487 LAPACKE_csysv_rook_work 4232
4.19.488 LAPACKE_csysv_work 4233
4.19.489 LAPACKE_csysvx 4233
4.19.490 LAPACKE_csysvx_work 4234
4.19.491 LAPACKE_csysvxx 4234
4.19.492 LAPACKE_csysvxx_work 4235
4.19.493 LAPACKE_csyswapr 4236
4.19.494 LAPACKE_csyswapr_work 4236
4.19.495 LAPACKE_csytrf 4237
4.19.496 LAPACKE_csytrf_aa 4237
4.19.497 LAPACKE_csytrf_aa_2stage 4238
4.19.498 LAPACKE_csytrf_aa_work 4238
4.19.499 LAPACKE_csytrf_rk 4239
4.19.500 LAPACKE_csytrf_rk_work 4239
4.19.501 LAPACKE_csytrf_rook 4240
4.19.502 LAPACKE_csytrf_rook_work 4240
4.19.503 LAPACKE_csytrf_work 4241
4.19.504 LAPACKE_csytri 4241
4.19.505 LAPACKE_csytri2 4242
4.19.506 LAPACKE_csytri2_work 4242
4.19.507 LAPACKE_csytri2x 4243
4.19.508 LAPACKE_csytri2x_work 4243
4.19.509 LAPACKE_csytri_3 4244
4.19.510 LAPACKE_csytri_3_work 4244
4.19.511 LAPACKE_csytri_work 4245
4.19.512 LAPACKE_csytrs 4245
4.19.513 LAPACKE_csytrs2 4246
4.19.514 LAPACKE_csytrs2_work 4246
4.19.515 LAPACKE_csytrs_3 4247
4.19.516 LAPACKE_csytrs_3_work 4248
4.19.517 LAPACKE_csytrs_aa 4248
4.19.518 LAPACKE_csytrs_aa_2stage 4249
4.19.519 LAPACKE_csytrs_aa_work 4249

4.19.520 LAPACKE_csytrs_rook 4250
4.19.521 LAPACKE_csytrs_rook_work 4250
4.19.522 LAPACKE_csytrs_work 4251
4.19.523 LAPACKE_ctbcon 4251
4.19.524 LAPACKE_ctbcon_work 4252
4.19.525 LAPACKE_ctbrfs 4253
4.19.526 LAPACKE_ctbrfs_work 4253
4.19.527 LAPACKE_ctbtrs 4254
4.19.528 LAPACKE_ctbtrs_work 4254
4.19.529 LAPACKE_ctfsm 4255
4.19.530 LAPACKE_ctfsm_work 4255
4.19.531 LAPACKE_ctftri 4256
4.19.532 LAPACKE_ctftri_work 4256
4.19.533 LAPACKE_ctfttp 4257
4.19.534 LAPACKE_ctfttp_work 4257
4.19.535 LAPACKE_ctfttr 4258
4.19.536 LAPACKE_ctfttr_work 4258
4.19.537 LAPACKE_ctgevc 4259
4.19.538 LAPACKE_ctgevc_work 4259
4.19.539 LAPACKE_ctgexc 4260
4.19.540 LAPACKE_ctgexc_work 4261
4.19.541 LAPACKE_ctgsen 4261
4.19.542 LAPACKE_ctgsen_work 4262
4.19.543 LAPACKE_ctgsja 4262
4.19.544 LAPACKE_ctgsja_work 4263
4.19.545 LAPACKE_ctgsna 4264
4.19.546 LAPACKE_ctgsna_work 4264
4.19.547 LAPACKE_ctgsyl 4265
4.19.548 LAPACKE_ctgsyl_work 4265
4.19.549 LAPACKE_ctpcon 4266
4.19.550 LAPACKE_ctpcon_work 4267
4.19.551 LAPACKE_ctpmqrt 4267
4.19.552 LAPACKE_ctpmqrt_work 4268
4.19.553 LAPACKE_ctpqrt 4268
4.19.554 LAPACKE_ctpqrt2 4269
4.19.555 LAPACKE_ctpqrt2_work 4269
4.19.556 LAPACKE_ctpqrt_work 4270
4.19.557 LAPACKE_ctprfb 4270
4.19.558 LAPACKE_ctprfb_work 4271
4.19.559 LAPACKE_ctprfs 4271
4.19.560 LAPACKE_ctprfs_work 4272
4.19.561 LAPACKE_ctptri 4273
4.19.562 LAPACKE_ctptri_work 4273
4.19.563 LAPACKE_ctptrs 4274
4.19.564 LAPACKE_ctptrs_work 4274
4.19.565 LAPACKE_ctpttf 4275
4.19.566 LAPACKE_ctpttf_work 4275
4.19.567 LAPACKE_ctpttr 4276
4.19.568 LAPACKE_ctpttr_work 4276
4.19.569 LAPACKE_ctrcon 4277
4.19.570 LAPACKE_ctrcon_work 4277
4.19.571 LAPACKE_ctrevc 4278
4.19.572 LAPACKE_ctrevc_work 4278
4.19.573 LAPACKE_ctrexc 4279
4.19.574 LAPACKE_ctrexc_work 4279
4.19.575 LAPACKE_ctrrfs 4280
4.19.576 LAPACKE_ctrrfs_work 4280
4.19.577 LAPACKE_ctrsen 4281

4.19.578 LAPACKE_ctrsen_work 4282
4.19.579 LAPACKE_ctrsna 4282
4.19.580 LAPACKE_ctrsna_work 4283
4.19.581 LAPACKE_ctrsyl 4283
4.19.582 LAPACKE_ctrsyl_work 4284
4.19.583 LAPACKE_ctrtri 4284
4.19.584 LAPACKE_ctrtri_work 4285
4.19.585 LAPACKE_ctrtrs 4285
4.19.586 LAPACKE_ctrtrs_work 4286
4.19.587 LAPACKE_ctrttf 4286
4.19.588 LAPACKE_ctrttf_work 4287
4.19.589 LAPACKE_ctrttp 4287
4.19.590 LAPACKE_ctrttp_work 4288
4.19.591 LAPACKE_ctzrzf 4288
4.19.592 LAPACKE_ctzrzf_work 4289
4.19.593 LAPACKE_cunbdb 4289
4.19.594 LAPACKE_cunbdb_work 4290
4.19.595 LAPACKE_cuncsd 4291
4.19.596 LAPACKE_cuncsd2by1 4291
4.19.597 LAPACKE_cuncsd2by1_work 4292
4.19.598 LAPACKE_cuncsd_work 4293
4.19.599 LAPACKE_cungbr 4293
4.19.600 LAPACKE_cungbr_work 4294
4.19.601 LAPACKE_cunghr 4294
4.19.602 LAPACKE_cunghr_work 4295
4.19.603 LAPACKE_cunglq 4295
4.19.604 LAPACKE_cunglq_work 4296
4.19.605 LAPACKE_cungql 4296
4.19.606 LAPACKE_cungql_work 4297
4.19.607 LAPACKE_cungqr 4297
4.19.608 LAPACKE_cungqr_work 4298
4.19.609 LAPACKE_cungrq 4298
4.19.610 LAPACKE_cungrq_work 4299
4.19.611 LAPACKE_cungtr 4299
4.19.612 LAPACKE_cungtr_work 4300
4.19.613 LAPACKE_cunmbr 4300
4.19.614 LAPACKE_cunmbr_work 4301
4.19.615 LAPACKE_cunmhr 4301
4.19.616 LAPACKE_cunmhr_work 4302
4.19.617 LAPACKE_cunmlq 4303
4.19.618 LAPACKE_cunmlq_work 4303
4.19.619 LAPACKE_cunmql 4304
4.19.620 LAPACKE_cunmql_work 4304
4.19.621 LAPACKE_cunmqr 4305
4.19.622 LAPACKE_cunmqr_work 4305
4.19.623 LAPACKE_cunmrq 4306
4.19.624 LAPACKE_cunmrq_work 4306
4.19.625 LAPACKE_cunmrz 4307
4.19.626 LAPACKE_cunmrz_work 4307
4.19.627 LAPACKE_cunmtr 4308
4.19.628 LAPACKE_cunmtr_work 4309
4.19.629 LAPACKE_cupgtr 4309
4.19.630 LAPACKE_cupgtr_work 4309
4.19.631 LAPACKE_cupmtr 4310
4.19.632 LAPACKE_cupmtr_work 4310
4.19.633 LAPACKE_dbbcsd 4311
4.19.634 LAPACKE_dbbcsd_work 4311
4.19.635 LAPACKE_dbdsdc 4312

4.19.636 LAPACKE_dbdsdc_work 4312
4.19.637 LAPACKE_dbdsqr 4313
4.19.638 LAPACKE_dbdsqr_work 4314
4.19.639 LAPACKE_dbdsvdx 4314
4.19.640 LAPACKE_dbdsvdx_work 4315
4.19.641 LAPACKE_ddisna 4315
4.19.642 LAPACKE_ddisna_work 4316
4.19.643 LAPACKE_dgbbrd 4316
4.19.644 LAPACKE_dgbbrd_work 4317
4.19.645 LAPACKE_dgbcon 4317
4.19.646 LAPACKE_dgbcon_work 4318
4.19.647 LAPACKE_dgbequ 4318
4.19.648 LAPACKE_dgbequ_work 4319
4.19.649 LAPACKE_dgbequb 4319
4.19.650 LAPACKE_dgbequb_work 4320
4.19.651 LAPACKE_dgbrfs 4320
4.19.652 LAPACKE_dgbrfs_work 4321
4.19.653 LAPACKE_dgbrfsx 4322
4.19.654 LAPACKE_dgbrfsx_work 4322
4.19.655 LAPACKE_dgbsv 4323
4.19.656 LAPACKE_dgbsv_work 4323
4.19.657 LAPACKE_dgbsvx 4324
4.19.658 LAPACKE_dgbsvx_work 4324
4.19.659 LAPACKE_dgbsvxx 4325
4.19.660 LAPACKE_dgbsvxx_work 4326
4.19.661 LAPACKE_dgbtrf 4326
4.19.662 LAPACKE_dgbtrf_work 4327
4.19.663 LAPACKE_dgbtrs 4327
4.19.664 LAPACKE_dgbtrs_work 4328
4.19.665 LAPACKE_dgebak 4328
4.19.666 LAPACKE_dgebak_work 4329
4.19.667 LAPACKE_dgebal 4329
4.19.668 LAPACKE_dgebal_work 4330
4.19.669 LAPACKE_dgebrd 4330
4.19.670 LAPACKE_dgebrd_work 4331
4.19.671 LAPACKE_dgecon 4331
4.19.672 LAPACKE_dgecon_work 4332
4.19.673 LAPACKE_dgeequ 4332
4.19.674 LAPACKE_dgeequ_work 4333
4.19.675 LAPACKE_dgeequb 4333
4.19.676 LAPACKE_dgeequb_work 4334
4.19.677 LAPACKE_dgees 4334
4.19.678 LAPACKE_dgees_work 4335
4.19.679 LAPACKE_dgeesx 4335
4.19.680 LAPACKE_dgeesx_work 4336
4.19.681 LAPACKE_dgeev 4337
4.19.682 LAPACKE_dgeev_work 4337
4.19.683 LAPACKE_dgeevx 4338
4.19.684 LAPACKE_dgeevx_work 4338
4.19.685 LAPACKE_dgehrd 4339
4.19.686 LAPACKE_dgehrd_work 4339
4.19.687 LAPACKE_dgejsv 4340
4.19.688 LAPACKE_dgejsv_work 4340
4.19.689 LAPACKE_dgelq 4341
4.19.690 LAPACKE_dgelq2 4341
4.19.691 LAPACKE_dgelq2_work 4342
4.19.692 LAPACKE_dgelq_work 4342
4.19.693 LAPACKE_dgelqf 4343

4.19.694 LAPACKE_dgelqf_work 4343
4.19.695 LAPACKE_dgels 4344
4.19.696 LAPACKE_dgels_work 4344
4.19.697 LAPACKE_dgelsd 4345
4.19.698 LAPACKE_dgelsd_work 4345
4.19.699 LAPACKE_dgelss 4346
4.19.700 LAPACKE_dgelss_work 4346
4.19.701 LAPACKE_dgelsy 4347
4.19.702 LAPACKE_dgelsy_work 4347
4.19.703 LAPACKE_dgemlq 4348
4.19.704 LAPACKE_dgemlq_work 4348
4.19.705 LAPACKE_dgemqr 4349
4.19.706 LAPACKE_dgemqr_work 4349
4.19.707 LAPACKE_dgemqrt 4350
4.19.708 LAPACKE_dgemqrt_work 4350
4.19.709 LAPACKE_dgeqlf 4351
4.19.710 LAPACKE_dgeqlf_work 4351
4.19.711 LAPACKE_dgeqp3 4352
4.19.712 LAPACKE_dgeqp3_work 4353
4.19.713 LAPACKE_dgeqpf 4353
4.19.714 LAPACKE_dgeqpf_work 4354
4.19.715 LAPACKE_dgeqr 4354
4.19.716 LAPACKE_dgeqr2 4355
4.19.717 LAPACKE_dgeqr2_work 4355
4.19.718 LAPACKE_dgeqr_work 4356
4.19.719 LAPACKE_dgeqrf 4356
4.19.720 LAPACKE_dgeqrf_work 4357
4.19.721 LAPACKE_dgeqrfp 4357
4.19.722 LAPACKE_dgeqrfp_work 4358
4.19.723 LAPACKE_dgeqrt 4358
4.19.724 LAPACKE_dgeqrt2 4359
4.19.725 LAPACKE_dgeqrt2_work 4359
4.19.726 LAPACKE_dgeqrt3 4360
4.19.727 LAPACKE_dgeqrt3_work 4360
4.19.728 LAPACKE_dgeqrt_work 4361
4.19.729 LAPACKE_dgerfs 4361
4.19.730 LAPACKE_dgerfs_work 4362
4.19.731 LAPACKE_dgerfsx 4362
4.19.732 LAPACKE_dgerfsx_work 4363
4.19.733 LAPACKE_dgerqf 4363
4.19.734 LAPACKE_dgerqf_work 4364
4.19.735 LAPACKE_dgesdd 4364
4.19.736 LAPACKE_dgesdd_work 4365
4.19.737 LAPACKE_dgesv 4365
4.19.738 LAPACKE_dgesv_work 4366
4.19.739 LAPACKE_dgesvd 4366
4.19.740 LAPACKE_dgesvd_work 4367
4.19.741 LAPACKE_dgesvdx 4367
4.19.742 LAPACKE_dgesvdx_work 4368
4.19.743 LAPACKE_dgesvj 4369
4.19.744 LAPACKE_dgesvj_work 4369
4.19.745 LAPACKE_dgesvx 4370
4.19.746 LAPACKE_dgesvx_work 4370
4.19.747 LAPACKE_dgesvxx 4371
4.19.748 LAPACKE_dgesvxx_work 4371
4.19.749 LAPACKE_dgetf2 4372
4.19.750 LAPACKE_dgetf2_work 4372
4.19.751 LAPACKE_dgetrf 4373

4.19.752 LAPACKE_dgetrf2 4374
4.19.753 LAPACKE_dgetrf2_work 4374
4.19.754 LAPACKE_dgetrf_work 4374
4.19.755 LAPACKE_dgetri 4375
4.19.756 LAPACKE_dgetri_work 4376
4.19.757 LAPACKE_dgetrs 4376
4.19.758 LAPACKE_dgetrs_work 4377
4.19.759 LAPACKE_dgetsls 4377
4.19.760 LAPACKE_dgetsls_work 4378
4.19.761 LAPACKE_dggbak 4378
4.19.762 LAPACKE_dggbak_work 4379
4.19.763 LAPACKE_dggbal 4379
4.19.764 LAPACKE_dggbal_work 4380
4.19.765 LAPACKE_dgges 4380
4.19.766 LAPACKE_dgges3 4381
4.19.767 LAPACKE_dgges3_work 4381
4.19.768 LAPACKE_dgges_work 4382
4.19.769 LAPACKE_dggesx 4382
4.19.770 LAPACKE_dggesx_work 4382
4.19.771 LAPACKE_dggev 4383
4.19.772 LAPACKE_dggev3 4384
4.19.773 LAPACKE_dggev3_work 4384
4.19.774 LAPACKE_dggev_work 4385
4.19.775 LAPACKE_dggevx 4385
4.19.776 LAPACKE_dggevx_work 4386
4.19.777 LAPACKE_dggglm 4386
4.19.778 LAPACKE_dggglm_work 4387
4.19.779 LAPACKE_dgghd3 4388
4.19.780 LAPACKE_dgghd3_work 4388
4.19.781 LAPACKE_dgghrd 4389
4.19.782 LAPACKE_dgghrd_work 4389
4.19.783 LAPACKE_dgglse 4390
4.19.784 LAPACKE_dgglse_work 4390
4.19.785 LAPACKE_dggqrf 4391
4.19.786 LAPACKE_dggqrf_work 4391
4.19.787 LAPACKE_dggrqf 4392
4.19.788 LAPACKE_dggrqf_work 4392
4.19.789 LAPACKE_dggsvd 4393
4.19.790 LAPACKE_dggsvd3 4393
4.19.791 LAPACKE_dggsvd3_work 4394
4.19.792 LAPACKE_dggsvd_work 4394
4.19.793 LAPACKE_dggsvp 4395
4.19.794 LAPACKE_dggsvp3 4396
4.19.795 LAPACKE_dggsvp3_work 4396
4.19.796 LAPACKE_dggsvp_work 4397
4.19.797 LAPACKE_dgtcon 4397
4.19.798 LAPACKE_dgtcon_work 4398
4.19.799 LAPACKE_dgtrfs 4399
4.19.800 LAPACKE_dgtrfs_work 4399
4.19.801 LAPACKE_dgtsv 4400
4.19.802 LAPACKE_dgtsv_work 4400
4.19.803 LAPACKE_dgtsvx 4401
4.19.804 LAPACKE_dgtsvx_work 4401
4.19.805 LAPACKE_dgttrf 4402
4.19.806 LAPACKE_dgttrf_work 4402
4.19.807 LAPACKE_dgttrs 4403
4.19.808 LAPACKE_dgttrs_work 4403
4.19.809 LAPACKE_dhgeqz 4404

4.19.810 LAPACKE_dhgeqz_work 4405
4.19.811 LAPACKE_dhsein 4405
4.19.812 LAPACKE_dhsein_work 4406
4.19.813 LAPACKE_dhseqr 4406
4.19.814 LAPACKE_dhseqr_work 4407
4.19.815 LAPACKE_dlacn2 4407
4.19.816 LAPACKE_dlacn2_work 4408
4.19.817 LAPACKE_dlacpy 4408
4.19.818 LAPACKE_dlacpy_work 4409
4.19.819 LAPACKE_dlag2s 4409
4.19.820 LAPACKE_dlag2s_work 4410
4.19.821 LAPACKE_dlagge 4410
4.19.822 LAPACKE_dlagge_work 4410
4.19.823 LAPACKE_dlagsy 4410
4.19.824 LAPACKE_dlagsy_work 4411
4.19.825 LAPACKE_dlamch 4411
4.19.826 LAPACKE_dlamch_work 4411
4.19.827 LAPACKE_dlange 4412
4.19.828 LAPACKE_dlange_work 4412
4.19.829 LAPACKE_dlansy 4413
4.19.830 LAPACKE_dlansy_work 4413
4.19.831 LAPACKE_dlantr 4414
4.19.832 LAPACKE_dlantr_work 4414
4.19.833 LAPACKE_dlapmr 4415
4.19.834 LAPACKE_dlapmr_work 4415
4.19.835 LAPACKE_dlapmt 4416
4.19.836 LAPACKE_dlapmt_work 4416
4.19.837 LAPACKE_dlapy2 4417
4.19.838 LAPACKE_dlapy2_work 4417
4.19.839 LAPACKE_dlapy3 4417
4.19.840 LAPACKE_dlapy3_work 4418
4.19.841 LAPACKE_dlarfb 4418
4.19.842 LAPACKE_dlarfb_work 4419
4.19.843 LAPACKE_dlarfg 4420
4.19.844 LAPACKE_dlarfg_work 4420
4.19.845 LAPACKE_dlarft 4421
4.19.846 LAPACKE_dlarft_work 4421
4.19.847 LAPACKE_dlarfx 4422
4.19.848 LAPACKE_dlarfx_work 4422
4.19.849 LAPACKE_dlarnv 4423
4.19.850 LAPACKE_dlarnv_work 4423
4.19.851 LAPACKE_dlartgp 4424
4.19.852 LAPACKE_dlartgp_work 4424
4.19.853 LAPACKE_dlartgs 4425
4.19.854 LAPACKE_dlartgs_work 4425
4.19.855 LAPACKE_dlascl 4426
4.19.856 LAPACKE_dlascl_work 4426
4.19.857 LAPACKE_dlaset 4427
4.19.858 LAPACKE_dlaset_work 4427
4.19.859 LAPACKE_dlasrt 4428
4.19.860 LAPACKE_dlasrt_work 4428
4.19.861 LAPACKE_dlassq 4428
4.19.862 LAPACKE_dlassq_work 4429
4.19.863 LAPACKE_dlaswp 4429
4.19.864 LAPACKE_dlaswp_work 4430
4.19.865 LAPACKE_dlatms 4431
4.19.866 LAPACKE_dlatms_work 4431
4.19.867 LAPACKE_dlauum 4431

4.19.868 LAPACKE_dlauum_work 4431
4.19.869 LAPACKE_dopgtr 4432
4.19.870 LAPACKE_dopgtr_work 4432
4.19.871 LAPACKE_dopmtr 4433
4.19.872 LAPACKE_dopmtr_work 4433
4.19.873 LAPACKE_dorbdb 4434
4.19.874 LAPACKE_dorbdb_work 4434
4.19.875 LAPACKE_dorcsd 4435
4.19.876 LAPACKE_dorcsd2by1 4435
4.19.877 LAPACKE_dorcsd2by1_work 4436
4.19.878 LAPACKE_dorcsd_work 4437
4.19.879 LAPACKE_dorgbr 4437
4.19.880 LAPACKE_dorgbr_work 4438
4.19.881 LAPACKE_dorghr 4438
4.19.882 LAPACKE_dorghr_work 4439
4.19.883 LAPACKE_dorglq 4439
4.19.884 LAPACKE_dorglq_work 4440
4.19.885 LAPACKE_dorgql 4440
4.19.886 LAPACKE_dorgql_work 4441
4.19.887 LAPACKE_dorgqr 4441
4.19.888 LAPACKE_dorgqr_work 4442
4.19.889 LAPACKE_dorgrq 4442
4.19.890 LAPACKE_dorgrq_work 4443
4.19.891 LAPACKE_dorgtr 4443
4.19.892 LAPACKE_dorgtr_work 4444
4.19.893 LAPACKE_dormbr 4444
4.19.894 LAPACKE_dormbr_work 4445
4.19.895 LAPACKE_dormhr 4445
4.19.896 LAPACKE_dormhr_work 4446
4.19.897 LAPACKE_dormlq 4446
4.19.898 LAPACKE_dormlq_work 4447
4.19.899 LAPACKE_dormql 4447
4.19.900 LAPACKE_dormql_work 4448
4.19.901 LAPACKE_dormqr 4448
4.19.902 LAPACKE_dormqr_work 4449
4.19.903 LAPACKE_dormrq 4449
4.19.904 LAPACKE_dormrq_work 4450
4.19.905 LAPACKE_dormrz 4450
4.19.906 LAPACKE_dormrz_work 4451
4.19.907 LAPACKE_dormtr 4451
4.19.908 LAPACKE_dormtr_work 4452
4.19.909 LAPACKE_dpbcon 4452
4.19.910 LAPACKE_dpbcon_work 4453
4.19.911 LAPACKE_dpbequ 4453
4.19.912 LAPACKE_dpbequ_work 4454
4.19.913 LAPACKE_dpbrfs 4454
4.19.914 LAPACKE_dpbrfs_work 4455
4.19.915 LAPACKE_dpbstf 4455
4.19.916 LAPACKE_dpbstf_work 4456
4.19.917 LAPACKE_dpbsv 4456
4.19.918 LAPACKE_dpbsv_work 4457
4.19.919 LAPACKE_dpbsvx 4457
4.19.920 LAPACKE_dpbsvx_work 4458
4.19.921 LAPACKE_dpbtrf 4459
4.19.922 LAPACKE_dpbtrf_work 4459
4.19.923 LAPACKE_dpbtrs 4460
4.19.924 LAPACKE_dpbtrs_work 4460
4.19.925 LAPACKE_dpftrf 4461

4.19.926 LAPACKE_dpftrf_work 4461
4.19.927 LAPACKE_dpftri 4462
4.19.928 LAPACKE_dpftri_work 4462
4.19.929 LAPACKE_dpftrs 4463
4.19.930 LAPACKE_dpftrs_work 4463
4.19.931 LAPACKE_dpocon 4464
4.19.932 LAPACKE_dpocon_work 4464
4.19.933 LAPACKE_dpoequ 4465
4.19.934 LAPACKE_dpoequ_work 4465
4.19.935 LAPACKE_dpoequb 4466
4.19.936 LAPACKE_dpoequb_work 4466
4.19.937 LAPACKE_dporfs 4467
4.19.938 LAPACKE_dporfs_work 4467
4.19.939 LAPACKE_dporfsx 4468
4.19.940 LAPACKE_dporfsx_work 4468
4.19.941 LAPACKE_dposv 4469
4.19.942 LAPACKE_dposv_work 4469
4.19.943 LAPACKE_dposvx 4470
4.19.944 LAPACKE_dposvx_work 4470
4.19.945 LAPACKE_dposvxx 4471
4.19.946 LAPACKE_dposvxx_work 4472
4.19.947 LAPACKE_dpotrf 4472
4.19.948 LAPACKE_dpotrf2 4473
4.19.949 LAPACKE_dpotrf2_work 4473
4.19.950 LAPACKE_dpotrf_work 4474
4.19.951 LAPACKE_dpotri 4474
4.19.952 LAPACKE_dpotri_work 4475
4.19.953 LAPACKE_dpotrs 4475
4.19.954 LAPACKE_dpotrs_work 4476
4.19.955 LAPACKE_dppcon 4476
4.19.956 LAPACKE_dppcon_work 4477
4.19.957 LAPACKE_dppequ 4477
4.19.958 LAPACKE_dppequ_work 4478
4.19.959 LAPACKE_dpprfs 4478
4.19.960 LAPACKE_dpprfs_work 4479
4.19.961 LAPACKE_dppsv 4479
4.19.962 LAPACKE_dppsv_work 4480
4.19.963 LAPACKE_dppsvx 4480
4.19.964 LAPACKE_dppsvx_work 4481
4.19.965 LAPACKE_dpptrf 4481
4.19.966 LAPACKE_dpptrf_work 4482
4.19.967 LAPACKE_dpptri 4482
4.19.968 LAPACKE_dpptri_work 4483
4.19.969 LAPACKE_dpptrs 4483
4.19.970 LAPACKE_dpptrs_work 4484
4.19.971 LAPACKE_dpstrf 4484
4.19.972 LAPACKE_dpstrf_work 4485
4.19.973 LAPACKE_dptcon 4485
4.19.974 LAPACKE_dptcon_work 4485
4.19.975 LAPACKE_dpteqr 4486
4.19.976 LAPACKE_dpteqr_work 4487
4.19.977 LAPACKE_dptrfs 4487
4.19.978 LAPACKE_dptrfs_work 4488
4.19.979 LAPACKE_dptsv 4488
4.19.980 LAPACKE_dptsv_work 4489
4.19.981 LAPACKE_dptsvx 4489
4.19.982 LAPACKE_dptsvx_work 4490
4.19.983 LAPACKE_dpttrf 4490

4.19.984 LAPACKE_dpttrf_work 4491
4.19.985 LAPACKE_dpttrs 4491
4.19.986 LAPACKE_dpttrs_work 4492
4.19.987 LAPACKE_dsbev 4492
4.19.988 LAPACKE_dsbev_2stage 4493
4.19.989 LAPACKE_dsbev_2stage_work 4493
4.19.990 LAPACKE_dsbev_work 4494
4.19.991 LAPACKE_dsbevd 4494
4.19.992 LAPACKE_dsbevd_2stage 4495
4.19.993 LAPACKE_dsbevd_2stage_work 4495
4.19.994 LAPACKE_dsbevd_work 4496
4.19.995 LAPACKE_dsbevx 4496
4.19.996 LAPACKE_dsbevx_2stage 4497
4.19.997 LAPACKE_dsbevx_2stage_work 4497
4.19.998 LAPACKE_dsbevx_work 4498
4.19.999 LAPACKE_dsbgst 4499
4.19.1000 LAPACKE_dsbgst_work 4499
4.19.1001 LAPACKE_dsbgv 4500
4.19.1002 LAPACKE_dsbgv_work 4500
4.19.1003 LAPACKE_dsbgvd 4501
4.19.1004 LAPACKE_dsbgvd_work 4501
4.19.1005 LAPACKE_dsbgvx 4502
4.19.1006 LAPACKE_dsbgvx_work 4502
4.19.1007 LAPACKE_dsbtrd 4503
4.19.1008 LAPACKE_dsbtrd_work 4503
4.19.1009 LAPACKE_dsfrk 4504
4.19.1010 LAPACKE_dsfrk_work 4504
4.19.1011 LAPACKE_dsgesv 4505
4.19.1012 LAPACKE_dsgesv_work 4505
4.19.1013 LAPACKE_dspcon 4506
4.19.1014 LAPACKE_dspcon_work 4506
4.19.1015 LAPACKE_dspev 4507
4.19.1016 LAPACKE_dspev_work 4507
4.19.1017 LAPACKE_dspevd 4508
4.19.1018 LAPACKE_dspevd_work 4508
4.19.1019 LAPACKE_dspevx 4509
4.19.1020 LAPACKE_dspevx_work 4509
4.19.1021 LAPACKE_dspgst 4510
4.19.1022 LAPACKE_dspgst_work 4510
4.19.1023 LAPACKE_dspgv 4511
4.19.1024 LAPACKE_dspgv_work 4511
4.19.1025 LAPACKE_dspgvd 4512
4.19.1026 LAPACKE_dspgvd_work 4512
4.19.1027 LAPACKE_dspgvx 4513
4.19.1028 LAPACKE_dspgvx_work 4513
4.19.1029 LAPACKE_dsposv 4514
4.19.1030 LAPACKE_dsposv_work 4514
4.19.1031 LAPACKE_dsprfs 4515
4.19.1032 LAPACKE_dsprfs_work 4516
4.19.1033 LAPACKE_dspsv 4516
4.19.1034 LAPACKE_dspsv_work 4517
4.19.1035 LAPACKE_dspsvx 4517
4.19.1036 LAPACKE_dspsvx_work 4518
4.19.1037 LAPACKE_dsptrd 4518
4.19.1038 LAPACKE_dsptrd_work 4519
4.19.1039 LAPACKE_dsptrf 4519
4.19.1040 LAPACKE_dsptrf_work 4520
4.19.1041 LAPACKE_dsptri 4520

4.19.1042 LAPACKE_dsptri_work 4521
4.19.1043 LAPACKE_dsptrs 4521
4.19.1044 LAPACKE_dsptrs_work 4522
4.19.1045 LAPACKE_dstebz 4522
4.19.1046 LAPACKE_dstebz_work 4523
4.19.1047 LAPACKE_dstedc 4523
4.19.1048 LAPACKE_dstedc_work 4524
4.19.1049 LAPACKE_dstegr 4524
4.19.1050 LAPACKE_dstegr_work 4525
4.19.1051 LAPACKE_dstein 4525
4.19.1052 LAPACKE_dstein_work 4526
4.19.1053 LAPACKE_dstemr 4526
4.19.1054 LAPACKE_dstemr_work 4527
4.19.1055 LAPACKE_dsteqr 4527
4.19.1056 LAPACKE_dsteqr_work 4528
4.19.1057 LAPACKE_dsterf 4528
4.19.1058 LAPACKE_dsterf_work 4529
4.19.1059 LAPACKE_dstev 4529
4.19.1060 LAPACKE_dstev_work 4530
4.19.1061 LAPACKE_dstevd 4530
4.19.1062 LAPACKE_dstevd_work 4531
4.19.1063 LAPACKE_dstevr 4531
4.19.1064 LAPACKE_dstevr_work 4532
4.19.1065 LAPACKE_dstevx 4532
4.19.1066 LAPACKE_dstevx_work 4533
4.19.1067 LAPACKE_dsycon 4533
4.19.1068 LAPACKE_dsycon_3 4534
4.19.1069 LAPACKE_dsycon_3_work 4534
4.19.1070 LAPACKE_dsycon_work 4535
4.19.1071 LAPACKE_dsyconv 4535
4.19.1072 LAPACKE_dsyconv_work 4536
4.19.1073 LAPACKE_dsyequb 4536
4.19.1074 LAPACKE_dsyequb_work 4537
4.19.1075 LAPACKE_dsyev 4537
4.19.1076 LAPACKE_dsyev_2stage 4538
4.19.1077 LAPACKE_dsyev_2stage_work 4538
4.19.1078 LAPACKE_dsyev_work 4539
4.19.1079 LAPACKE_dsyevd 4539
4.19.1080 LAPACKE_dsyevd_2stage 4540
4.19.1081 LAPACKE_dsyevd_2stage_work 4540
4.19.1082 LAPACKE_dsyevd_work 4541
4.19.1083 LAPACKE_dsyevr 4541
4.19.1084 LAPACKE_dsyevr_2stage 4541
4.19.1085 LAPACKE_dsyevr_2stage_work 4542
4.19.1086 LAPACKE_dsyevr_work 4543
4.19.1087 LAPACKE_dsyevx 4543
4.19.1088 LAPACKE_dsyevx_2stage 4544
4.19.1089 LAPACKE_dsyevx_2stage_work 4544
4.19.1090 LAPACKE_dsyevx_work 4545
4.19.1091 LAPACKE_dsygst 4545
4.19.1092 LAPACKE_dsygst_work 4546
4.19.1093 LAPACKE_dsygv 4546
4.19.1094 LAPACKE_dsygv_2stage 4547
4.19.1095 LAPACKE_dsygv_2stage_work 4547
4.19.1096 LAPACKE_dsygv_work 4548
4.19.1097 LAPACKE_dsygvd 4548
4.19.1098 LAPACKE_dsygvd_work 4549
4.19.1099 LAPACKE_dsygvx 4549

4.19.1100 LAPACKE_dsygvx_work 4550
4.19.1101 LAPACKE_dsyrfs 4551
4.19.1102 LAPACKE_dsyrfs_work 4551
4.19.1103 LAPACKE_dsyrfsx 4552
4.19.1104 LAPACKE_dsyrfsx_work 4552
4.19.1105 LAPACKE_dsysv 4553
4.19.1106 LAPACKE_dsysv_aa 4553
4.19.1107 LAPACKE_dsysv_aa_2stage 4554
4.19.1108 LAPACKE_dsysv_aa_work 4554
4.19.1109 LAPACKE_dsysv_rk 4555
4.19.1110 LAPACKE_dsysv_rk_work 4555
4.19.1111 LAPACKE_dsysv_rook 4556
4.19.1112 LAPACKE_dsysv_rook_work 4557
4.19.1113 LAPACKE_dsysv_work 4557
4.19.1114 LAPACKE_dsysvx 4558
4.19.1115 LAPACKE_dsysvx_work 4558
4.19.1116 LAPACKE_dsysvxx 4559
4.19.1117 LAPACKE_dsysvxx_work 4559
4.19.1118 LAPACKE_dsyswapr 4560
4.19.1119 LAPACKE_dsyswapr_work 4561
4.19.1120 LAPACKE_dsytrd 4561
4.19.1121 LAPACKE_dsytrd_work 4562
4.19.1122 LAPACKE_dsytrf 4562
4.19.1123 LAPACKE_dsytrf_aa 4563
4.19.1124 LAPACKE_dsytrf_aa_2stage 4563
4.19.1125 LAPACKE_dsytrf_aa_work 4564
4.19.1126 LAPACKE_dsytrf_rk 4564
4.19.1127 LAPACKE_dsytrf_rk_work 4565
4.19.1128 LAPACKE_dsytrf_rook 4565
4.19.1129 LAPACKE_dsytrf_rook_work 4566
4.19.1130 LAPACKE_dsytrf_work 4566
4.19.1131 LAPACKE_dsytri 4567
4.19.1132 LAPACKE_dsytri2 4567
4.19.1133 LAPACKE_dsytri2_work 4567
4.19.1134 LAPACKE_dsytri2x 4568
4.19.1135 LAPACKE_dsytri2x_work 4569
4.19.1136 LAPACKE_dsytri_3 4569
4.19.1137 LAPACKE_dsytri_3_work 4570
4.19.1138 LAPACKE_dsytri_work 4570
4.19.1139 LAPACKE_dsytrs 4571
4.19.1140 LAPACKE_dsytrs2 4571
4.19.1141 LAPACKE_dsytrs2_work 4572
4.19.1142 LAPACKE_dsytrs_3 4572
4.19.1143 LAPACKE_dsytrs_3_work 4573
4.19.1144 LAPACKE_dsytrs_aa 4573
4.19.1145 LAPACKE_dsytrs_aa_2stage 4574
4.19.1146 LAPACKE_dsytrs_aa_work 4574
4.19.1147 LAPACKE_dsytrs_rook 4575
4.19.1148 LAPACKE_dsytrs_rook_work 4575
4.19.1149 LAPACKE_dsytrs_work 4576
4.19.1150 LAPACKE_dtbcon 4576
4.19.1151 LAPACKE_dtbcon_work 4577
4.19.1152 LAPACKE_dtbrfs 4577
4.19.1153 LAPACKE_dtbrfs_work 4578
4.19.1154 LAPACKE_dtbtrs 4578
4.19.1155 LAPACKE_dtbtrs_work 4579
4.19.1156 LAPACKE_dtfsm 4579
4.19.1157 LAPACKE_dtfsm_work 4580

4.19.1158 LAPACKE_dtftri 4580
4.19.1159 LAPACKE_dtftri_work 4581
4.19.1160 LAPACKE_dtfttp 4581
4.19.1161 LAPACKE_dtfttp_work 4582
4.19.1162 LAPACKE_dtfttr 4582
4.19.1163 LAPACKE_dtfttr_work 4583
4.19.1164 LAPACKE_dtgevc 4583
4.19.1165 LAPACKE_dtgevc_work 4584
4.19.1166 LAPACKE_dtgexc 4584
4.19.1167 LAPACKE_dtgexc_work 4585
4.19.1168 LAPACKE_dtgsen 4586
4.19.1169 LAPACKE_dtgsen_work 4586
4.19.1170 LAPACKE_dtgsja 4587
4.19.1171 LAPACKE_dtgsja_work 4587
4.19.1172 LAPACKE_dtgsna 4588
4.19.1173 LAPACKE_dtgsna_work 4588
4.19.1174 LAPACKE_dtgsyl 4589
4.19.1175 LAPACKE_dtgsyl_work 4590
4.19.1176 LAPACKE_dtpcon 4590
4.19.1177 LAPACKE_dtpcon_work 4591
4.19.1178 LAPACKE_dtpmqrt 4591
4.19.1179 LAPACKE_dtpmqrt_work 4592
4.19.1180 LAPACKE_dtpqrt 4592
4.19.1181 LAPACKE_dtpqrt2 4593
4.19.1182 LAPACKE_dtpqrt2_work 4593
4.19.1183 LAPACKE_dtpqrt_work 4594
4.19.1184 LAPACKE_dtprfb 4594
4.19.1185 LAPACKE_dtprfb_work 4595
4.19.1186 LAPACKE_dtprfs 4595
4.19.1187 LAPACKE_dtprfs_work 4596
4.19.1188 LAPACKE_dtptri 4597
4.19.1189 LAPACKE_dtptri_work 4597
4.19.1190 LAPACKE_dtptrs 4598
4.19.1191 LAPACKE_dtptrs_work 4598
4.19.1192 LAPACKE_dtpttf 4599
4.19.1193 LAPACKE_dtpttf_work 4599
4.19.1194 LAPACKE_dtpttr 4599
4.19.1195 LAPACKE_dtpttr_work 4600
4.19.1196 LAPACKE_dtrcon 4600
4.19.1197 LAPACKE_dtrcon_work 4601
4.19.1198 LAPACKE_dtrevc 4602
4.19.1199 LAPACKE_dtrevc_work 4602
4.19.1200 LAPACKE_dtrexc 4603
4.19.1201 LAPACKE_dtrexc_work 4603
4.19.1202 LAPACKE_dtrrfs 4604
4.19.1203 LAPACKE_dtrrfs_work 4604
4.19.1204 LAPACKE_dtrsen 4605
4.19.1205 LAPACKE_dtrsen_work 4605
4.19.1206 LAPACKE_dtrsna 4606
4.19.1207 LAPACKE_dtrsna_work 4606
4.19.1208 LAPACKE_dtrsyl 4607
4.19.1209 LAPACKE_dtrsyl_work 4608
4.19.1210 LAPACKE_dtrtri 4608
4.19.1211 LAPACKE_dtrtri_work 4609
4.19.1212 LAPACKE_dtrtrs 4609
4.19.1213 LAPACKE_dtrtrs_work 4610
4.19.1214 LAPACKE_dtrttf 4610
4.19.1215 LAPACKE_dtrttf_work 4611

4.19.1216 LAPACKE_dtrttp 4611
4.19.1217 LAPACKE_dtrttp_work 4612
4.19.1218 LAPACKE_dtzrzf 4612
4.19.1219 LAPACKE_dtzrzf_work 4613
4.19.1220 LAPACKE_ilaver 4613
4.19.1221 LAPACKE_sbbcsd 4613
4.19.1222 LAPACKE_sbbcsd_work 4614
4.19.1223 LAPACKE_sbdsdc 4615
4.19.1224 LAPACKE_sbdsdc_work 4615
4.19.1225 LAPACKE_sbdsqr 4616
4.19.1226 LAPACKE_sbdsqr_work 4616
4.19.1227 LAPACKE_sbdsvdx 4617
4.19.1228 LAPACKE_sbdsvdx_work 4617
4.19.1229 LAPACKE_sdisna 4618
4.19.1230 LAPACKE_sdisna_work 4618
4.19.1231 LAPACKE_sgbbrd 4619
4.19.1232 LAPACKE_sgbbrd_work 4619
4.19.1233 LAPACKE_sgbcon 4620
4.19.1234 LAPACKE_sgbcon_work 4620
4.19.1235 LAPACKE_sgbequ 4621
4.19.1236 LAPACKE_sgbequ_work 4621
4.19.1237 LAPACKE_sgbequb 4622
4.19.1238 LAPACKE_sgbequb_work 4622
4.19.1239 LAPACKE_sgbrfs 4623
4.19.1240 LAPACKE_sgbrfs_work 4623
4.19.1241 LAPACKE_sgbrfsx 4624
4.19.1242 LAPACKE_sgbrfsx_work 4625
4.19.1243 LAPACKE_sgbsv 4625
4.19.1244 LAPACKE_sgbsv_work 4626
4.19.1245 LAPACKE_sgbsvx 4626
4.19.1246 LAPACKE_sgbsvx_work 4627
4.19.1247 LAPACKE_sgbsvxx 4627
4.19.1248 LAPACKE_sgbsvxx_work 4628
4.19.1249 LAPACKE_sgbtrf 4629
4.19.1250 LAPACKE_sgbtrf_work 4629
4.19.1251 LAPACKE_sgbtrs 4630
4.19.1252 LAPACKE_sgbtrs_work 4630
4.19.1253 LAPACKE_sgebak 4631
4.19.1254 LAPACKE_sgebak_work 4631
4.19.1255 LAPACKE_sgebal 4631
4.19.1256 LAPACKE_sgebal_work 4632
4.19.1257 LAPACKE_sgebrd 4632
4.19.1258 LAPACKE_sgebrd_work 4633
4.19.1259 LAPACKE_sgecon 4633
4.19.1260 LAPACKE_sgecon_work 4634
4.19.1261 LAPACKE_sgeequ 4634
4.19.1262 LAPACKE_sgeequ_work 4635
4.19.1263 LAPACKE_sgeequb 4635
4.19.1264 LAPACKE_sgeequb_work 4636
4.19.1265 LAPACKE_sgees 4636
4.19.1266 LAPACKE_sgees_work 4637
4.19.1267 LAPACKE_sgeesx 4637
4.19.1268 LAPACKE_sgeesx_work 4638
4.19.1269 LAPACKE_sgeev 4639
4.19.1270 LAPACKE_sgeev_work 4639
4.19.1271 LAPACKE_sgeevx 4640
4.19.1272 LAPACKE_sgeevx_work 4640
4.19.1273 LAPACKE_sgehrd 4641

4.19.1274 LAPACKE_sgehrd_work 4641
4.19.1275 LAPACKE_sgejsv 4642
4.19.1276 LAPACKE_sgejsv_work 4642
4.19.1277 LAPACKE_sgelq 4643
4.19.1278 LAPACKE_sgelq2 4643
4.19.1279 LAPACKE_sgelq2_work 4644
4.19.1280 LAPACKE_sgelq_work 4644
4.19.1281 LAPACKE_sgelqf 4645
4.19.1282 LAPACKE_sgelqf_work 4645
4.19.1283 LAPACKE_sgels 4646
4.19.1284 LAPACKE_sgels_work 4646
4.19.1285 LAPACKE_sgelsd 4647
4.19.1286 LAPACKE_sgelsd_work 4647
4.19.1287 LAPACKE_sgelss 4648
4.19.1288 LAPACKE_sgelss_work 4648
4.19.1289 LAPACKE_sgelsy 4649
4.19.1290 LAPACKE_sgelsy_work 4649
4.19.1291 LAPACKE_sgemlq 4650
4.19.1292 LAPACKE_sgemlq_work 4650
4.19.1293 LAPACKE_sgemqr 4651
4.19.1294 LAPACKE_sgemqr_work 4652
4.19.1295 LAPACKE_sgemqrt 4652
4.19.1296 LAPACKE_sgemqrt_work 4653
4.19.1297 LAPACKE_sgeqlf 4653
4.19.1298 LAPACKE_sgeqlf_work 4654
4.19.1299 LAPACKE_sgeqp3 4654
4.19.1300 LAPACKE_sgeqp3_work 4655
4.19.1301 LAPACKE_sgeqpf 4655
4.19.1302 LAPACKE_sgeqpf_work 4656
4.19.1303 LAPACKE_sgeqr 4656
4.19.1304 LAPACKE_sgeqr2 4657
4.19.1305 LAPACKE_sgeqr2_work 4657
4.19.1306 LAPACKE_sgeqr_work 4658
4.19.1307 LAPACKE_sgeqrf 4658
4.19.1308 LAPACKE_sgeqrf_work 4659
4.19.1309 LAPACKE_sgeqrfp 4659
4.19.1310 LAPACKE_sgeqrfp_work 4660
4.19.1311 LAPACKE_sgeqrt 4660
4.19.1312 LAPACKE_sgeqrt2 4661
4.19.1313 LAPACKE_sgeqrt2_work 4661
4.19.1314 LAPACKE_sgeqrt3 4662
4.19.1315 LAPACKE_sgeqrt3_work 4662
4.19.1316 LAPACKE_sgeqrt_work 4663
4.19.1317 LAPACKE_sgerfs 4663
4.19.1318 LAPACKE_sgerfs_work 4664
4.19.1319 LAPACKE_sgerfsx 4664
4.19.1320 LAPACKE_sgerfsx_work 4665
4.19.1321 LAPACKE_sgerqf 4665
4.19.1322 LAPACKE_sgerqf_work 4666
4.19.1323 LAPACKE_sgesdd 4667
4.19.1324 LAPACKE_sgesdd_work 4667
4.19.1325 LAPACKE_sgesv 4668
4.19.1326 LAPACKE_sgesv_work 4668
4.19.1327 LAPACKE_sgesvd 4669
4.19.1328 LAPACKE_sgesvd_work 4669
4.19.1329 LAPACKE_sgesvdx 4670
4.19.1330 LAPACKE_sgesvdx_work 4670
4.19.1331 LAPACKE_sgesvj 4671

4.19.1332 LAPACKE_sgesvj_work 4671
4.19.1333 LAPACKE_sgesvx 4672
4.19.1334 LAPACKE_sgesvx_work 4672
4.19.1335 LAPACKE_sgesvxx 4673
4.19.1336 LAPACKE_sgesvxx_work 4673
4.19.1337 LAPACKE_sgetf2 4674
4.19.1338 LAPACKE_sgetf2_work 4675
4.19.1339 LAPACKE_sgetrf 4675
4.19.1340 LAPACKE_sgetrf2 4676
4.19.1341 LAPACKE_sgetrf2_work 4676
4.19.1342 LAPACKE_sgetrf_work 4677
4.19.1343 LAPACKE_sgetri 4677
4.19.1344 LAPACKE_sgetri_work 4678
4.19.1345 LAPACKE_sgetrs 4678
4.19.1346 LAPACKE_sgetrs_work 4679
4.19.1347 LAPACKE_sgetsls 4679
4.19.1348 LAPACKE_sgetsls_work 4680
4.19.1349 LAPACKE_sggbak 4680
4.19.1350 LAPACKE_sggbak_work 4681
4.19.1351 LAPACKE_sggbal 4681
4.19.1352 LAPACKE_sggbal_work 4682
4.19.1353 LAPACKE_sgges 4682
4.19.1354 LAPACKE_sgges3 4683
4.19.1355 LAPACKE_sgges3_work 4683
4.19.1356 LAPACKE_sgges_work 4684
4.19.1357 LAPACKE_sggesx 4684
4.19.1358 LAPACKE_sggesx_work 4685
4.19.1359 LAPACKE_sggev 4686
4.19.1360 LAPACKE_sggev3 4686
4.19.1361 LAPACKE_sggev3_work 4687
4.19.1362 LAPACKE_sggev_work 4687
4.19.1363 LAPACKE_sggevx 4688
4.19.1364 LAPACKE_sggevx_work 4688
4.19.1365 LAPACKE_sggglm 4689
4.19.1366 LAPACKE_sggglm_work 4689
4.19.1367 LAPACKE_sgghd3 4690
4.19.1368 LAPACKE_sgghd3_work 4690
4.19.1369 LAPACKE_sgghrd 4691
4.19.1370 LAPACKE_sgghrd_work 4691
4.19.1371 LAPACKE_sgglse 4692
4.19.1372 LAPACKE_sgglse_work 4693
4.19.1373 LAPACKE_sggqrf 4693
4.19.1374 LAPACKE_sggqrf_work 4694
4.19.1375 LAPACKE_sggrqf 4694
4.19.1376 LAPACKE_sggrqf_work 4695
4.19.1377 LAPACKE_sggsvd 4695
4.19.1378 LAPACKE_sggsvd3 4696
4.19.1379 LAPACKE_sggsvd3_work 4696
4.19.1380 LAPACKE_sggsvd_work 4697
4.19.1381 LAPACKE_sggsvp 4697
4.19.1382 LAPACKE_sggsvp3 4698
4.19.1383 LAPACKE_sggsvp3_work 4699
4.19.1384 LAPACKE_sggsvp_work 4699
4.19.1385 LAPACKE_sgtcon 4700
4.19.1386 LAPACKE_sgtcon_work 4700
4.19.1387 LAPACKE_sgtrfs 4701
4.19.1388 LAPACKE_sgtrfs_work 4701
4.19.1389 LAPACKE_sgtsv 4702

4.19.1390 LAPACKE_sgtsv_work 4702
4.19.1391 LAPACKE_sgtsvx 4703
4.19.1392 LAPACKE_sgtsvx_work 4703
4.19.1393 LAPACKE_sgttrf 4704
4.19.1394 LAPACKE_sgttrf_work 4705
4.19.1395 LAPACKE_sgttrs 4705
4.19.1396 LAPACKE_sgttrs_work 4706
4.19.1397 LAPACKE_shgeqz 4706
4.19.1398 LAPACKE_shgeqz_work 4707
4.19.1399 LAPACKE_shsein 4707
4.19.1400 LAPACKE_shsein_work 4708
4.19.1401 LAPACKE_shseqr 4708
4.19.1402 LAPACKE_shseqr_work 4709
4.19.1403 LAPACKE_slacn2 4709
4.19.1404 LAPACKE_slacn2_work 4710
4.19.1405 LAPACKE_slacpy 4710
4.19.1406 LAPACKE_slacpy_work 4711
4.19.1407 LAPACKE_slag2d 4711
4.19.1408 LAPACKE_slag2d_work 4712
4.19.1409 LAPACKE_slagge 4712
4.19.1410 LAPACKE_slagge_work 4712
4.19.1411 LAPACKE_slagsy 4713
4.19.1412 LAPACKE_slagsy_work 4713
4.19.1413 LAPACKE_slamch 4713
4.19.1414 LAPACKE_slamch_work 4713
4.19.1415 LAPACKE_slange 4714
4.19.1416 LAPACKE_slange_work 4714
4.19.1417 LAPACKE_slansy 4715
4.19.1418 LAPACKE_slansy_work 4715
4.19.1419 LAPACKE_slantr 4716
4.19.1420 LAPACKE_slantr_work 4716
4.19.1421 LAPACKE_slapmr 4717
4.19.1422 LAPACKE_slapmr_work 4717
4.19.1423 LAPACKE_slapmt 4718
4.19.1424 LAPACKE_slapmt_work 4718
4.19.1425 LAPACKE_slapy2 4718
4.19.1426 LAPACKE_slapy2_work 4719
4.19.1427 LAPACKE_slapy3 4719
4.19.1428 LAPACKE_slapy3_work 4720
4.19.1429 LAPACKE_slarfb 4720
4.19.1430 LAPACKE_slarfb_work 4721
4.19.1431 LAPACKE_slarfg 4721
4.19.1432 LAPACKE_slarfg_work 4722
4.19.1433 LAPACKE_slarft 4722
4.19.1434 LAPACKE_slarft_work 4723
4.19.1435 LAPACKE_slarfx 4723
4.19.1436 LAPACKE_slarfx_work 4724
4.19.1437 LAPACKE_slarnv 4724
4.19.1438 LAPACKE_slarnv_work 4725
4.19.1439 LAPACKE_slartgp 4725
4.19.1440 LAPACKE_slartgp_work 4726
4.19.1441 LAPACKE_slartgs 4726
4.19.1442 LAPACKE_slartgs_work 4727
4.19.1443 LAPACKE_slascl 4727
4.19.1444 LAPACKE_slascl_work 4728
4.19.1445 LAPACKE_slaset 4728
4.19.1446 LAPACKE_slaset_work 4729
4.19.1447 LAPACKE_slasrt 4729

4.19.1448 LAPACKE_slasrt_work 4730
4.19.1449 LAPACKE_slassq 4730
4.19.1450 LAPACKE_slassq_work 4731
4.19.1451 LAPACKE_slaswp 4731
4.19.1452 LAPACKE_slaswp_work 4732
4.19.1453 LAPACKE_slatms 4732
4.19.1454 LAPACKE_slatms_work 4732
4.19.1455 LAPACKE_slauum 4733
4.19.1456 LAPACKE_slauum_work 4733
4.19.1457 LAPACKE_sopgtr 4734
4.19.1458 LAPACKE_sopgtr_work 4734
4.19.1459 LAPACKE_sopmtr 4735
4.19.1460 LAPACKE_sopmtr_work 4735
4.19.1461 LAPACKE_sorbdb 4736
4.19.1462 LAPACKE_sorbdb_work 4736
4.19.1463 LAPACKE_sorcsd 4737
4.19.1464 LAPACKE_sorcsd2by1 4737
4.19.1465 LAPACKE_sorcsd2by1_work 4738
4.19.1466 LAPACKE_sorcsd_work 4738
4.19.1467 LAPACKE_sorgbr 4739
4.19.1468 LAPACKE_sorgbr_work 4739
4.19.1469 LAPACKE_sorghr 4740
4.19.1470 LAPACKE_sorghr_work 4741
4.19.1471 LAPACKE_sorglq 4741
4.19.1472 LAPACKE_sorglq_work 4742
4.19.1473 LAPACKE_sorgql 4742
4.19.1474 LAPACKE_sorgql_work 4743
4.19.1475 LAPACKE_sorgqr 4743
4.19.1476 LAPACKE_sorgqr_work 4744
4.19.1477 LAPACKE_sorgrq 4744
4.19.1478 LAPACKE_sorgrq_work 4745
4.19.1479 LAPACKE_sorgtr 4745
4.19.1480 LAPACKE_sorgtr_work 4746
4.19.1481 LAPACKE_sormbr 4746
4.19.1482 LAPACKE_sormbr_work 4747
4.19.1483 LAPACKE_sormhr 4747
4.19.1484 LAPACKE_sormhr_work 4748
4.19.1485 LAPACKE_sormlq 4748
4.19.1486 LAPACKE_sormlq_work 4749
4.19.1487 LAPACKE_sormql 4749
4.19.1488 LAPACKE_sormql_work 4750
4.19.1489 LAPACKE_sormqr 4750
4.19.1490 LAPACKE_sormqr_work 4751
4.19.1491 LAPACKE_sormrq 4751
4.19.1492 LAPACKE_sormrq_work 4752
4.19.1493 LAPACKE_sormrz 4752
4.19.1494 LAPACKE_sormrz_work 4753
4.19.1495 LAPACKE_sormtr 4753
4.19.1496 LAPACKE_sormtr_work 4754
4.19.1497 LAPACKE_spbcon 4754
4.19.1498 LAPACKE_spbcon_work 4755
4.19.1499 LAPACKE_spbequ 4755
4.19.1500 LAPACKE_spbequ_work 4756
4.19.1501 LAPACKE_spbrfs 4756
4.19.1502 LAPACKE_spbrfs_work 4757
4.19.1503 LAPACKE_spbstf 4757
4.19.1504 LAPACKE_spbstf_work 4758
4.19.1505 LAPACKE_spbsv 4758

4.19.1506 LAPACKE_spbsv_work 4759
4.19.1507 LAPACKE_spbsvx 4759
4.19.1508 LAPACKE_spbsvx_work 4760
4.19.1509 LAPACKE_spbtrf 4760
4.19.1510 LAPACKE_spbtrf_work 4761
4.19.1511 LAPACKE_spbtrs 4761
4.19.1512 LAPACKE_spbtrs_work 4762
4.19.1513 LAPACKE_spftrf 4762
4.19.1514 LAPACKE_spftrf_work 4763
4.19.1515 LAPACKE_spftri 4763
4.19.1516 LAPACKE_spftri_work 4764
4.19.1517 LAPACKE_spftrs 4764
4.19.1518 LAPACKE_spftrs_work 4765
4.19.1519 LAPACKE_spocon 4765
4.19.1520 LAPACKE_spocon_work 4766
4.19.1521 LAPACKE_spoequ 4766
4.19.1522 LAPACKE_spoequ_work 4767
4.19.1523 LAPACKE_spoequb 4767
4.19.1524 LAPACKE_spoequb_work 4768
4.19.1525 LAPACKE_sporfs 4768
4.19.1526 LAPACKE_sporfs_work 4769
4.19.1527 LAPACKE_sporfsx 4769
4.19.1528 LAPACKE_sporfsx_work 4770
4.19.1529 LAPACKE_sposv 4770
4.19.1530 LAPACKE_sposv_work 4771
4.19.1531 LAPACKE_sposvx 4771
4.19.1532 LAPACKE_sposvx_work 4772
4.19.1533 LAPACKE_sposvxx 4772
4.19.1534 LAPACKE_sposvxx_work 4773
4.19.1535 LAPACKE_spotrf 4774
4.19.1536 LAPACKE_spotrf2 4774
4.19.1537 LAPACKE_spotrf2_work 4775
4.19.1538 LAPACKE_spotrf_work 4775
4.19.1539 LAPACKE_spotri 4776
4.19.1540 LAPACKE_spotri_work 4776
4.19.1541 LAPACKE_spotrs 4777
4.19.1542 LAPACKE_spotrs_work 4777
4.19.1543 LAPACKE_sppcon 4778
4.19.1544 LAPACKE_sppcon_work 4778
4.19.1545 LAPACKE_sppequ 4779
4.19.1546 LAPACKE_sppequ_work 4779
4.19.1547 LAPACKE_spprfs 4780
4.19.1548 LAPACKE_spprfs_work 4780
4.19.1549 LAPACKE_sppsv 4781
4.19.1550 LAPACKE_sppsv_work 4781
4.19.1551 LAPACKE_sppsvx 4782
4.19.1552 LAPACKE_sppsvx_work 4782
4.19.1553 LAPACKE_spptrf 4783
4.19.1554 LAPACKE_spptrf_work 4783
4.19.1555 LAPACKE_spptri 4784
4.19.1556 LAPACKE_spptri_work 4784
4.19.1557 LAPACKE_spptrs 4785
4.19.1558 LAPACKE_spptrs_work 4785
4.19.1559 LAPACKE_spstrf 4786
4.19.1560 LAPACKE_spstrf_work 4786
4.19.1561 LAPACKE_sptcon 4787
4.19.1562 LAPACKE_sptcon_work 4787
4.19.1563 LAPACKE_spteqr 4788

4.19.1564 LAPACKE_spteqr_work 4788
4.19.1565 LAPACKE_sptrfs 4789
4.19.1566 LAPACKE_sptrfs_work 4789
4.19.1567 LAPACKE_sptsv 4790
4.19.1568 LAPACKE_sptsv_work 4790
4.19.1569 LAPACKE_sptsvx 4791
4.19.1570 LAPACKE_sptsvx_work 4791
4.19.1571 LAPACKE_spttrf 4792
4.19.1572 LAPACKE_spttrf_work 4792
4.19.1573 LAPACKE_spttrs 4793
4.19.1574 LAPACKE_spttrs_work 4793
4.19.1575 LAPACKE_ssbev 4794
4.19.1576 LAPACKE_ssbev_2stage 4794
4.19.1577 LAPACKE_ssbev_2stage_work 4795
4.19.1578 LAPACKE_ssbev_work 4795
4.19.1579 LAPACKE_ssbevd 4796
4.19.1580 LAPACKE_ssbevd_2stage 4796
4.19.1581 LAPACKE_ssbevd_2stage_work 4797
4.19.1582 LAPACKE_ssbevd_work 4797
4.19.1583 LAPACKE_ssbevx 4798
4.19.1584 LAPACKE_ssbevx_2stage 4798
4.19.1585 LAPACKE_ssbevx_2stage_work 4799
4.19.1586 LAPACKE_ssbevx_work 4799
4.19.1587 LAPACKE_ssbgst 4800
4.19.1588 LAPACKE_ssbgst_work 4801
4.19.1589 LAPACKE_ssbgv 4801
4.19.1590 LAPACKE_ssbgv_work 4802
4.19.1591 LAPACKE_ssbgvd 4802
4.19.1592 LAPACKE_ssbgvd_work 4803
4.19.1593 LAPACKE_ssbgvx 4803
4.19.1594 LAPACKE_ssbgvx_work 4804
4.19.1595 LAPACKE_ssbtrd 4804
4.19.1596 LAPACKE_ssbtrd_work 4805
4.19.1597 LAPACKE_ssfrk 4805
4.19.1598 LAPACKE_ssfrk_work 4806
4.19.1599 LAPACKE_sspcon 4806
4.19.1600 LAPACKE_sspcon_work 4807
4.19.1601 LAPACKE_sspev 4807
4.19.1602 LAPACKE_sspev_work 4808
4.19.1603 LAPACKE_sspevd 4808
4.19.1604 LAPACKE_sspevd_work 4809
4.19.1605 LAPACKE_sspevx 4809
4.19.1606 LAPACKE_sspevx_work 4810
4.19.1607 LAPACKE_sspgst 4810
4.19.1608 LAPACKE_sspgst_work 4811
4.19.1609 LAPACKE_sspgv 4811
4.19.1610 LAPACKE_sspgv_work 4812
4.19.1611 LAPACKE_sspgvd 4812
4.19.1612 LAPACKE_sspgvd_work 4813
4.19.1613 LAPACKE_sspgvx 4813
4.19.1614 LAPACKE_sspgvx_work 4814
4.19.1615 LAPACKE_ssprfs 4814
4.19.1616 LAPACKE_ssprfs_work 4815
4.19.1617 LAPACKE_sspsv 4815
4.19.1618 LAPACKE_sspsv_work 4816
4.19.1619 LAPACKE_sspsvx 4816
4.19.1620 LAPACKE_sspsvx_work 4817
4.19.1621 LAPACKE_ssptrd 4817

4.19.1622 LAPACKE_ssptrd_work 4818
4.19.1623 LAPACKE_ssptrf 4818
4.19.1624 LAPACKE_ssptrf_work 4819
4.19.1625 LAPACKE_ssptri 4819
4.19.1626 LAPACKE_ssptri_work 4820
4.19.1627 LAPACKE_ssptrs 4820
4.19.1628 LAPACKE_ssptrs_work 4821
4.19.1629 LAPACKE_sstebz 4821
4.19.1630 LAPACKE_sstebz_work 4822
4.19.1631 LAPACKE_sstedc 4822
4.19.1632 LAPACKE_sstedc_work 4823
4.19.1633 LAPACKE_sstegr 4823
4.19.1634 LAPACKE_sstegr_work 4824
4.19.1635 LAPACKE_sstein 4824
4.19.1636 LAPACKE_sstein_work 4825
4.19.1637 LAPACKE_sstemr 4826
4.19.1638 LAPACKE_sstemr_work 4826
4.19.1639 LAPACKE_ssteqr 4827
4.19.1640 LAPACKE_ssteqr_work 4827
4.19.1641 LAPACKE_ssterf 4828
4.19.1642 LAPACKE_ssterf_work 4828
4.19.1643 LAPACKE_sstev 4829
4.19.1644 LAPACKE_sstev_work 4829
4.19.1645 LAPACKE_sstevd 4830
4.19.1646 LAPACKE_sstevd_work 4830
4.19.1647 LAPACKE_sstevr 4831
4.19.1648 LAPACKE_sstevr_work 4831
4.19.1649 LAPACKE_sstevx 4832
4.19.1650 LAPACKE_sstevx_work 4832
4.19.1651 LAPACKE_ssycon 4833
4.19.1652 LAPACKE_ssycon_3 4833
4.19.1653 LAPACKE_ssycon_3_work 4834
4.19.1654 LAPACKE_ssycon_work 4834
4.19.1655 LAPACKE_ssyconv 4835
4.19.1656 LAPACKE_ssyconv_work 4835
4.19.1657 LAPACKE_ssyequb 4836
4.19.1658 LAPACKE_ssyequb_work 4836
4.19.1659 LAPACKE_ssyev 4837
4.19.1660 LAPACKE_ssyev_2stage 4837
4.19.1661 LAPACKE_ssyev_2stage_work 4838
4.19.1662 LAPACKE_ssyev_work 4838
4.19.1663 LAPACKE_ssyevd 4839
4.19.1664 LAPACKE_ssyevd_2stage 4839
4.19.1665 LAPACKE_ssyevd_2stage_work 4840
4.19.1666 LAPACKE_ssyevd_work 4840
4.19.1667 LAPACKE_ssyevr 4841
4.19.1668 LAPACKE_ssyevr_2stage 4841
4.19.1669 LAPACKE_ssyevr_2stage_work 4842
4.19.1670 LAPACKE_ssyevr_work 4843
4.19.1671 LAPACKE_ssyevx 4843
4.19.1672 LAPACKE_ssyevx_2stage 4844
4.19.1673 LAPACKE_ssyevx_2stage_work 4844
4.19.1674 LAPACKE_ssyevx_work 4845
4.19.1675 LAPACKE_ssygst 4845
4.19.1676 LAPACKE_ssygst_work 4846
4.19.1677 LAPACKE_ssygv 4846
4.19.1678 LAPACKE_ssygv_2stage 4847
4.19.1679 LAPACKE_ssygv_2stage_work 4847

4.19.1680 LAPACKE_ssygv_work 4847
4.19.1681 LAPACKE_ssygvd 4848
4.19.1682 LAPACKE_ssygvd_work 4848
4.19.1683 LAPACKE_ssygvx 4849
4.19.1684 LAPACKE_ssygvx_work 4850
4.19.1685 LAPACKE_ssyrfs 4850
4.19.1686 LAPACKE_ssyrfs_work 4851
4.19.1687 LAPACKE_ssyrfsx 4851
4.19.1688 LAPACKE_ssyrfsx_work 4852
4.19.1689 LAPACKE_ssysv 4852
4.19.1690 LAPACKE_ssysv_aa 4853
4.19.1691 LAPACKE_ssysv_aa_2stage 4854
4.19.1692 LAPACKE_ssysv_aa_work 4854
4.19.1693 LAPACKE_ssysv_rk 4855
4.19.1694 LAPACKE_ssysv_rk_work 4855
4.19.1695 LAPACKE_ssysv_rook 4856
4.19.1696 LAPACKE_ssysv_rook_work 4856
4.19.1697 LAPACKE_ssysv_work 4857
4.19.1698 LAPACKE_ssysvx 4857
4.19.1699 LAPACKE_ssysvx_work 4858
4.19.1700 LAPACKE_ssysvxx 4858
4.19.1701 LAPACKE_ssysvxx_work 4859
4.19.1702 LAPACKE_ssyswapr 4859
4.19.1703 LAPACKE_ssyswapr_work 4860
4.19.1704 LAPACKE_ssytrd 4861
4.19.1705 LAPACKE_ssytrd_work 4861
4.19.1706 LAPACKE_ssytrf 4862
4.19.1707 LAPACKE_ssytrf_aa 4862
4.19.1708 LAPACKE_ssytrf_aa_2stage 4863
4.19.1709 LAPACKE_ssytrf_aa_work 4863
4.19.1710 LAPACKE_ssytrf_rk 4864
4.19.1711 LAPACKE_ssytrf_rk_work 4864
4.19.1712 LAPACKE_ssytrf_rook 4865
4.19.1713 LAPACKE_ssytrf_rook_work 4865
4.19.1714 LAPACKE_ssytrf_work 4866
4.19.1715 LAPACKE_ssytri 4866
4.19.1716 LAPACKE_ssytri2 4867
4.19.1717 LAPACKE_ssytri2_work 4867
4.19.1718 LAPACKE_ssytri2x 4868
4.19.1719 LAPACKE_ssytri2x_work 4868
4.19.1720 LAPACKE_ssytri_3 4869
4.19.1721 LAPACKE_ssytri_3_work 4869
4.19.1722 LAPACKE_ssytri_work 4870
4.19.1723 LAPACKE_ssytrs 4870
4.19.1724 LAPACKE_ssytrs2 4871
4.19.1725 LAPACKE_ssytrs2_work 4871
4.19.1726 LAPACKE_ssytrs_3 4872
4.19.1727 LAPACKE_ssytrs_3_work 4872
4.19.1728 LAPACKE_ssytrs_aa 4873
4.19.1729 LAPACKE_ssytrs_aa_2stage 4873
4.19.1730 LAPACKE_ssytrs_aa_work 4874
4.19.1731 LAPACKE_ssytrs_rook 4874
4.19.1732 LAPACKE_ssytrs_rook_work 4875
4.19.1733 LAPACKE_ssytrs_work 4875
4.19.1734 LAPACKE_stbcon 4876
4.19.1735 LAPACKE_stbcon_work 4876
4.19.1736 LAPACKE_stbrfs 4877
4.19.1737 LAPACKE_stbrfs_work 4877

4.19.1738 LAPACKE_stbtrs 4878
4.19.1739 LAPACKE_stbtrs_work 4878
4.19.1740 LAPACKE_stfsm 4879
4.19.1741 LAPACKE_stfsm_work 4879
4.19.1742 LAPACKE_stftri 4880
4.19.1743 LAPACKE_stftri_work 4880
4.19.1744 LAPACKE_stfttp 4881
4.19.1745 LAPACKE_stfttp_work 4881
4.19.1746 LAPACKE_stfttr 4882
4.19.1747 LAPACKE_stfttr_work 4882
4.19.1748 LAPACKE_stgevc 4883
4.19.1749 LAPACKE_stgevc_work 4883
4.19.1750 LAPACKE_stgexc 4884
4.19.1751 LAPACKE_stgexc_work 4885
4.19.1752 LAPACKE_stgsen 4885
4.19.1753 LAPACKE_stgsen_work 4886
4.19.1754 LAPACKE_stgsja 4886
4.19.1755 LAPACKE_stgsja_work 4887
4.19.1756 LAPACKE_stgsna 4887
4.19.1757 LAPACKE_stgsna_work 4888
4.19.1758 LAPACKE_stgsyl 4889
4.19.1759 LAPACKE_stgsyl_work 4889
4.19.1760 LAPACKE_stpcon 4890
4.19.1761 LAPACKE_stpcon_work 4890
4.19.1762 LAPACKE_stpmqrt 4891
4.19.1763 LAPACKE_stpmqrt_work 4891
4.19.1764 LAPACKE_stpqrt 4892
4.19.1765 LAPACKE_stpqrt2 4892
4.19.1766 LAPACKE_stpqrt2_work 4893
4.19.1767 LAPACKE_stpqrt_work 4893
4.19.1768 LAPACKE_stprfb 4894
4.19.1769 LAPACKE_stprfb_work 4894
4.19.1770 LAPACKE_stprfs 4895
4.19.1771 LAPACKE_stprfs_work 4896
4.19.1772 LAPACKE_stptri 4896
4.19.1773 LAPACKE_stptri_work 4897
4.19.1774 LAPACKE_stptrs 4897
4.19.1775 LAPACKE_stptrs_work 4898
4.19.1776 LAPACKE_stpttf 4898
4.19.1777 LAPACKE_stpttf_work 4899
4.19.1778 LAPACKE_stpttr 4899
4.19.1779 LAPACKE_stpttr_work 4900
4.19.1780 LAPACKE_strcon 4900
4.19.1781 LAPACKE_strcon_work 4901
4.19.1782 LAPACKE_strevc 4901
4.19.1783 LAPACKE_strevc_work 4902
4.19.1784 LAPACKE_strexc 4902
4.19.1785 LAPACKE_strexc_work 4903
4.19.1786 LAPACKE_strrfs 4903
4.19.1787 LAPACKE_strrfs_work 4904
4.19.1788 LAPACKE_strsen 4904
4.19.1789 LAPACKE_strsen_work 4905
4.19.1790 LAPACKE_strsna 4905
4.19.1791 LAPACKE_strsna_work 4906
4.19.1792 LAPACKE_strsyl 4906
4.19.1793 LAPACKE_strsyl_work 4907
4.19.1794 LAPACKE_strtri 4907
4.19.1795 LAPACKE_strtri_work 4908

4.19.1796 LAPACKE_strtrs 4908
4.19.1797 LAPACKE_strtrs_work 4909
4.19.1798 LAPACKE_strttf 4909
4.19.1799 LAPACKE_strttf_work 4910
4.19.1800 LAPACKE_strttp 4910
4.19.1801 LAPACKE_strttp_work 4911
4.19.1802 LAPACKE_stzrzf 4911
4.19.1803 LAPACKE_stzrzf_work 4912
4.19.1804 LAPACKE_zbbcsd 4912
4.19.1805 LAPACKE_zbbcsd_work 4913
4.19.1806 LAPACKE_zbdsqr 4914
4.19.1807 LAPACKE_zbdsqr_work 4914
4.19.1808 LAPACKE_zcgesv 4915
4.19.1809 LAPACKE_zcgesv_work 4915
4.19.1810 LAPACKE_zcposv 4916
4.19.1811 LAPACKE_zcposv_work 4916
4.19.1812 LAPACKE_zgbbrd 4917
4.19.1813 LAPACKE_zgbbrd_work 4917
4.19.1814 LAPACKE_zgbcon 4918
4.19.1815 LAPACKE_zgbcon_work 4919
4.19.1816 LAPACKE_zgbequ 4919
4.19.1817 LAPACKE_zgbequ_work 4920
4.19.1818 LAPACKE_zgbequb 4920
4.19.1819 LAPACKE_zgbequb_work 4921
4.19.1820 LAPACKE_zgbrfs 4921
4.19.1821 LAPACKE_zgbrfs_work 4922
4.19.1822 LAPACKE_zgbrfsx 4922
4.19.1823 LAPACKE_zgbrfsx_work 4923
4.19.1824 LAPACKE_zgbsv 4924
4.19.1825 LAPACKE_zgbsv_work 4924
4.19.1826 LAPACKE_zgbsvx 4925
4.19.1827 LAPACKE_zgbsvx_work 4925
4.19.1828 LAPACKE_zgbsvxx 4926
4.19.1829 LAPACKE_zgbsvxx_work 4927
4.19.1830 LAPACKE_zgbtrf 4927
4.19.1831 LAPACKE_zgbtrf_work 4928
4.19.1832 LAPACKE_zgbtrs 4928
4.19.1833 LAPACKE_zgbtrs_work 4929
4.19.1834 LAPACKE_zgebak 4930
4.19.1835 LAPACKE_zgebak_work 4930
4.19.1836 LAPACKE_zgebal 4931
4.19.1837 LAPACKE_zgebal_work 4931
4.19.1838 LAPACKE_zgebrd 4932
4.19.1839 LAPACKE_zgebrd_work 4932
4.19.1840 LAPACKE_zgecon 4933
4.19.1841 LAPACKE_zgecon_work 4933
4.19.1842 LAPACKE_zgeequ 4934
4.19.1843 LAPACKE_zgeequ_work 4934
4.19.1844 LAPACKE_zgeequb 4935
4.19.1845 LAPACKE_zgeequb_work 4935
4.19.1846 LAPACKE_zgees 4936
4.19.1847 LAPACKE_zgees_work 4936
4.19.1848 LAPACKE_zgeesx 4937
4.19.1849 LAPACKE_zgeesx_work 4937
4.19.1850 LAPACKE_zgeev 4938
4.19.1851 LAPACKE_zgeev_work 4938
4.19.1852 LAPACKE_zgeevx 4939
4.19.1853 LAPACKE_zgeevx_work 4940

4.19.1854 LAPACKE_zgehrd 4940
4.19.1855 LAPACKE_zgehrd_work 4941
4.19.1856 LAPACKE_zgejsv 4941
4.19.1857 LAPACKE_zgejsv_work 4942
4.19.1858 LAPACKE_zgelq 4942
4.19.1859 LAPACKE_zgelq2 4943
4.19.1860 LAPACKE_zgelq2_work 4943
4.19.1861 LAPACKE_zgelq_work 4944
4.19.1862 LAPACKE_zgelqf 4944
4.19.1863 LAPACKE_zgelqf_work 4945
4.19.1864 LAPACKE_zgels 4945
4.19.1865 LAPACKE_zgels_work 4946
4.19.1866 LAPACKE_zgelsd 4946
4.19.1867 LAPACKE_zgelsd_work 4947
4.19.1868 LAPACKE_zgelss 4947
4.19.1869 LAPACKE_zgelss_work 4948
4.19.1870 LAPACKE_zgelsy 4949
4.19.1871 LAPACKE_zgelsy_work 4949
4.19.1872 LAPACKE_zgemlq 4950
4.19.1873 LAPACKE_zgemlq_work 4950
4.19.1874 LAPACKE_zgemqr 4951
4.19.1875 LAPACKE_zgemqr_work 4951
4.19.1876 LAPACKE_zgemqrt 4952
4.19.1877 LAPACKE_zgemqrt_work 4952
4.19.1878 LAPACKE_zgeqlf 4953
4.19.1879 LAPACKE_zgeqlf_work 4954
4.19.1880 LAPACKE_zgeqp3 4954
4.19.1881 LAPACKE_zgeqp3_work 4955
4.19.1882 LAPACKE_zgeqpf 4955
4.19.1883 LAPACKE_zgeqpf_work 4956
4.19.1884 LAPACKE_zgeqr 4956
4.19.1885 LAPACKE_zgeqr2 4957
4.19.1886 LAPACKE_zgeqr2_work 4957
4.19.1887 LAPACKE_zgeqr_work 4958
4.19.1888 LAPACKE_zgeqrf 4958
4.19.1889 LAPACKE_zgeqrf_work 4959
4.19.1890 LAPACKE_zgeqrfp 4959
4.19.1891 LAPACKE_zgeqrfp_work 4960
4.19.1892 LAPACKE_zgeqrt 4960
4.19.1893 LAPACKE_zgeqrt2 4961
4.19.1894 LAPACKE_zgeqrt2_work 4961
4.19.1895 LAPACKE_zgeqrt3 4962
4.19.1896 LAPACKE_zgeqrt3_work 4962
4.19.1897 LAPACKE_zgeqrt_work 4963
4.19.1898 LAPACKE_zgerfs 4963
4.19.1899 LAPACKE_zgerfs_work 4964
4.19.1900 LAPACKE_zgerfsx 4964
4.19.1901 LAPACKE_zgerfsx_work 4965
4.19.1902 LAPACKE_zgerqf 4966
4.19.1903 LAPACKE_zgerqf_work 4966
4.19.1904 LAPACKE_zgesdd 4967
4.19.1905 LAPACKE_zgesdd_work 4967
4.19.1906 LAPACKE_zgesv 4968
4.19.1907 LAPACKE_zgesv_work 4968
4.19.1908 LAPACKE_zgesvd 4969
4.19.1909 LAPACKE_zgesvd_work 4969
4.19.1910 LAPACKE_zgesvdx 4970
4.19.1911 LAPACKE_zgesvdx_work 4970

4.19.1912 LAPACKE_zgesvj 4971
4.19.1913 LAPACKE_zgesvj_work 4972
4.19.1914 LAPACKE_zgesvx 4972
4.19.1915 LAPACKE_zgesvx_work 4973
4.19.1916 LAPACKE_zgesvxx 4973
4.19.1917 LAPACKE_zgesvxx_work 4974
4.19.1918 LAPACKE_zgetf2 4975
4.19.1919 LAPACKE_zgetf2_work 4975
4.19.1920 LAPACKE_zgetrf 4976
4.19.1921 LAPACKE_zgetrf2 4976
4.19.1922 LAPACKE_zgetrf2_work 4977
4.19.1923 LAPACKE_zgetrf_work 4977
4.19.1924 LAPACKE_zgetri 4978
4.19.1925 LAPACKE_zgetri_work 4978
4.19.1926 LAPACKE_zgetrs 4979
4.19.1927 LAPACKE_zgetrs_work 4979
4.19.1928 LAPACKE_zgetsls 4980
4.19.1929 LAPACKE_zgetsls_work 4980
4.19.1930 LAPACKE_zggbak 4981
4.19.1931 LAPACKE_zggbak_work 4981
4.19.1932 LAPACKE_zggbal 4982
4.19.1933 LAPACKE_zggbal_work 4982
4.19.1934 LAPACKE_zgges 4983
4.19.1935 LAPACKE_zgges3 4983
4.19.1936 LAPACKE_zgges3_work 4984
4.19.1937 LAPACKE_zgges_work 4985
4.19.1938 LAPACKE_zggesx 4985
4.19.1939 LAPACKE_zggesx_work 4986
4.19.1940 LAPACKE_zggev 4986
4.19.1941 LAPACKE_zggev3 4987
4.19.1942 LAPACKE_zggev3_work 4988
4.19.1943 LAPACKE_zggev_work 4988
4.19.1944 LAPACKE_zggevx 4989
4.19.1945 LAPACKE_zggevx_work 4989
4.19.1946 LAPACKE_zggglm 4990
4.19.1947 LAPACKE_zggglm_work 4991
4.19.1948 LAPACKE_zgghd3 4991
4.19.1949 LAPACKE_zgghd3_work 4992
4.19.1950 LAPACKE_zgghrd 4992
4.19.1951 LAPACKE_zgghrd_work 4993
4.19.1952 LAPACKE_zgglse 4994
4.19.1953 LAPACKE_zgglse_work 4994
4.19.1954 LAPACKE_zggqrf 4995
4.19.1955 LAPACKE_zggqrf_work 4995
4.19.1956 LAPACKE_zggrqf 4996
4.19.1957 LAPACKE_zggrqf_work 4996
4.19.1958 LAPACKE_zggsvd 4997
4.19.1959 LAPACKE_zggsvd3 4997
4.19.1960 LAPACKE_zggsvd3_work 4998
4.19.1961 LAPACKE_zggsvd_work 4999
4.19.1962 LAPACKE_zggsvp 4999
4.19.1963 LAPACKE_zggsvp3 5000
4.19.1964 LAPACKE_zggsvp3_work 5001
4.19.1965 LAPACKE_zggsvp_work 5001
4.19.1966 LAPACKE_zgtcon 5002
4.19.1967 LAPACKE_zgtcon_work 5002
4.19.1968 LAPACKE_zgtrfs 5003
4.19.1969 LAPACKE_zgtrfs_work 5004

4.19.1970 LAPACKE_zgtsv 5004
4.19.1971 LAPACKE_zgtsv_work 5005
4.19.1972 LAPACKE_zgtsvx 5005
4.19.1973 LAPACKE_zgtsvx_work 5006
4.19.1974 LAPACKE_zgttrf 5007
4.19.1975 LAPACKE_zgttrf_work 5007
4.19.1976 LAPACKE_zgttrs 5008
4.19.1977 LAPACKE_zgttrs_work 5008
4.19.1978 LAPACKE_zhbev 5009
4.19.1979 LAPACKE_zhbev_2stage 5009
4.19.1980 LAPACKE_zhbev_2stage_work 5010
4.19.1981 LAPACKE_zhbev_work 5010
4.19.1982 LAPACKE_zhbevd 5011
4.19.1983 LAPACKE_zhbevd_2stage 5012
4.19.1984 LAPACKE_zhbevd_2stage_work 5012
4.19.1985 LAPACKE_zhbevd_work 5013
4.19.1986 LAPACKE_zhbevx 5013
4.19.1987 LAPACKE_zhbevx_2stage 5014
4.19.1988 LAPACKE_zhbevx_2stage_work 5014
4.19.1989 LAPACKE_zhbevx_work 5015
4.19.1990 LAPACKE_zhbgst 5016
4.19.1991 LAPACKE_zhbgst_work 5016
4.19.1992 LAPACKE_zhbgv 5017
4.19.1993 LAPACKE_zhbgv_work 5017
4.19.1994 LAPACKE_zhbgvd 5018
4.19.1995 LAPACKE_zhbgvd_work 5018
4.19.1996 LAPACKE_zhbgvx 5019
4.19.1997 LAPACKE_zhbgvx_work 5020
4.19.1998 LAPACKE_zhbtrd 5020
4.19.1999 LAPACKE_zhbtrd_work 5021
4.19.2000 LAPACKE_zhecon 5021
4.19.2001 LAPACKE_zhecon_3 5022
4.19.2002 LAPACKE_zhecon_3_work 5022
4.19.2003 LAPACKE_zhecon_work 5023
4.19.2004 LAPACKE_zheequb 5023
4.19.2005 LAPACKE_zheequb_work 5024
4.19.2006 LAPACKE_zheev 5024
4.19.2007 LAPACKE_zheev_2stage 5025
4.19.2008 LAPACKE_zheev_2stage_work 5025
4.19.2009 LAPACKE_zheev_work 5026
4.19.2010 LAPACKE_zheevd 5026
4.19.2011 LAPACKE_zheevd_2stage 5027
4.19.2012 LAPACKE_zheevd_2stage_work 5027
4.19.2013 LAPACKE_zheevd_work 5028
4.19.2014 LAPACKE_zheevr 5028
4.19.2015 LAPACKE_zheevr_2stage 5029
4.19.2016 LAPACKE_zheevr_2stage_work 5029
4.19.2017 LAPACKE_zheevr_work 5030
4.19.2018 LAPACKE_zheevx 5031
4.19.2019 LAPACKE_zheevx_2stage 5031
4.19.2020 LAPACKE_zheevx_2stage_work 5032
4.19.2021 LAPACKE_zheevx_work 5032
4.19.2022 LAPACKE_zhegst 5033
4.19.2023 LAPACKE_zhegst_work 5034
4.19.2024 LAPACKE_zhegv 5034
4.19.2025 LAPACKE_zhegv_2stage 5035
4.19.2026 LAPACKE_zhegv_2stage_work 5035
4.19.2027 LAPACKE_zhegv_work 5036

4.19.2028 LAPACKE_zhegvd 5036
4.19.2029 LAPACKE_zhegvd_work 5037
4.19.2030 LAPACKE_zhegvx 5037
4.19.2031 LAPACKE_zhegvx_work 5038
4.19.2032 LAPACKE_zherfs 5038
4.19.2033 LAPACKE_zherfs_work 5039
4.19.2034 LAPACKE_zherfsx 5040
4.19.2035 LAPACKE_zherfsx_work 5040
4.19.2036 LAPACKE_zhesv 5041
4.19.2037 LAPACKE_zhesv_aa 5041
4.19.2038 LAPACKE_zhesv_aa_2stage 5042
4.19.2039 LAPACKE_zhesv_aa_work 5042
4.19.2040 LAPACKE_zhesv_rk 5043
4.19.2041 LAPACKE_zhesv_rk_work 5044
4.19.2042 LAPACKE_zhesv_work 5044
4.19.2043 LAPACKE_zhesvx 5045
4.19.2044 LAPACKE_zhesvx_work 5045
4.19.2045 LAPACKE_zhesvxx 5046
4.19.2046 LAPACKE_zhesvxx_work 5046
4.19.2047 LAPACKE_zheswapr 5047
4.19.2048 LAPACKE_zheswapr_work 5048
4.19.2049 LAPACKE_zhetrd 5048
4.19.2050 LAPACKE_zhetrd_work 5048
4.19.2051 LAPACKE_zhetrf 5049
4.19.2052 LAPACKE_zhetrf_aa 5049
4.19.2053 LAPACKE_zhetrf_aa_2stage 5050
4.19.2054 LAPACKE_zhetrf_aa_work 5050
4.19.2055 LAPACKE_zhetrf_rk 5051
4.19.2056 LAPACKE_zhetrf_rk_work 5051
4.19.2057 LAPACKE_zhetrf_rook 5052
4.19.2058 LAPACKE_zhetrf_rook_work 5052
4.19.2059 LAPACKE_zhetrf_work 5053
4.19.2060 LAPACKE_zhetri 5053
4.19.2061 LAPACKE_zhetri2 5054
4.19.2062 LAPACKE_zhetri2_work 5054
4.19.2063 LAPACKE_zhetri2x 5055
4.19.2064 LAPACKE_zhetri2x_work 5055
4.19.2065 LAPACKE_zhetri_3 5056
4.19.2066 LAPACKE_zhetri_3_work 5056
4.19.2067 LAPACKE_zhetri_work 5057
4.19.2068 LAPACKE_zhetrs 5057
4.19.2069 LAPACKE_zhetrs2 5058
4.19.2070 LAPACKE_zhetrs2_work 5058
4.19.2071 LAPACKE_zhetrs_3 5059
4.19.2072 LAPACKE_zhetrs_3_work 5059
4.19.2073 LAPACKE_zhetrs_aa 5060
4.19.2074 LAPACKE_zhetrs_aa_2stage 5060
4.19.2075 LAPACKE_zhetrs_aa_work 5061
4.19.2076 LAPACKE_zhetrs_rook 5062
4.19.2077 LAPACKE_zhetrs_rook_work 5062
4.19.2078 LAPACKE_zhetrs_work 5063
4.19.2079 LAPACKE_zhfrk 5063
4.19.2080 LAPACKE_zhfrk_work 5064
4.19.2081 LAPACKE_zhgeqz 5064
4.19.2082 LAPACKE_zhgeqz_work 5065
4.19.2083 LAPACKE_zhpcon 5065
4.19.2084 LAPACKE_zhpcon_work 5066
4.19.2085 LAPACKE_zhpev 5066

4.19.2086 LAPACKE_zhpev_work 5067
4.19.2087 LAPACKE_zhpevd 5067
4.19.2088 LAPACKE_zhpevd_work 5068
4.19.2089 LAPACKE_zhpevx 5068
4.19.2090 LAPACKE_zhpevx_work 5069
4.19.2091 LAPACKE_zhpgst 5070
4.19.2092 LAPACKE_zhpgst_work 5070
4.19.2093 LAPACKE_zhpgv 5071
4.19.2094 LAPACKE_zhpgv_work 5071
4.19.2095 LAPACKE_zhpgvd 5072
4.19.2096 LAPACKE_zhpgvd_work 5072
4.19.2097 LAPACKE_zhpgvx 5073
4.19.2098 LAPACKE_zhpgvx_work 5073
4.19.2099 LAPACKE_zhprfs 5074
4.19.2100 LAPACKE_zhprfs_work 5074
4.19.2101 LAPACKE_zhpsv 5075
4.19.2102 LAPACKE_zhpsv_work 5075
4.19.2103 LAPACKE_zhpsvx 5076
4.19.2104 LAPACKE_zhpsvx_work 5077
4.19.2105 LAPACKE_zhptrd 5077
4.19.2106 LAPACKE_zhptrd_work 5078
4.19.2107 LAPACKE_zhptrf 5078
4.19.2108 LAPACKE_zhptrf_work 5079
4.19.2109 LAPACKE_zhptri 5079
4.19.2110 LAPACKE_zhptri_work 5080
4.19.2111 LAPACKE_zhptrs 5080
4.19.2112 LAPACKE_zhptrs_work 5081
4.19.2113 LAPACKE_zhsein 5081
4.19.2114 LAPACKE_zhsein_work 5082
4.19.2115 LAPACKE_zhseqr 5082
4.19.2116 LAPACKE_zhseqr_work 5083
4.19.2117 LAPACKE_zlacgv 5083
4.19.2118 LAPACKE_zlacgv_work 5084
4.19.2119 LAPACKE_zlacn2 5084
4.19.2120 LAPACKE_zlacn2_work 5085
4.19.2121 LAPACKE_zlacp2 5085
4.19.2122 LAPACKE_zlacp2_work 5086
4.19.2123 LAPACKE_zlacpy 5086
4.19.2124 LAPACKE_zlacpy_work 5087
4.19.2125 LAPACKE_zlacrm 5087
4.19.2126 LAPACKE_zlacrm_work 5088
4.19.2127 LAPACKE_zlag2c 5088
4.19.2128 LAPACKE_zlag2c_work 5089
4.19.2129 LAPACKE_zlagge 5089
4.19.2130 LAPACKE_zlagge_work 5089
4.19.2131 LAPACKE_zlaghe 5090
4.19.2132 LAPACKE_zlaghe_work 5090
4.19.2133 LAPACKE_zlagsy 5090
4.19.2134 LAPACKE_zlagsy_work 5090
4.19.2135 LAPACKE_zlange 5090
4.19.2136 LAPACKE_zlange_work 5091
4.19.2137 LAPACKE_zlanhe 5091
4.19.2138 LAPACKE_zlanhe_work 5092
4.19.2139 LAPACKE_zlansy 5092
4.19.2140 LAPACKE_zlansy_work 5093
4.19.2141 LAPACKE_zlantr 5093
4.19.2142 LAPACKE_zlantr_work 5094
4.19.2143 LAPACKE_zlapmr 5094

4.19.2144 LAPACKE_zlapmr_work 5095
4.19.2145 LAPACKE_zlapmt 5095
4.19.2146 LAPACKE_zlapmt_work 5096
4.19.2147 LAPACKE_zlarcm 5096
4.19.2148 LAPACKE_zlarcm_work 5097
4.19.2149 LAPACKE_zlarfb 5097
4.19.2150 LAPACKE_zlarfb_work 5098
4.19.2151 LAPACKE_zlarfg 5098
4.19.2152 LAPACKE_zlarfg_work 5099
4.19.2153 LAPACKE_zlarft 5099
4.19.2154 LAPACKE_zlarft_work 5100
4.19.2155 LAPACKE_zlarfx 5100
4.19.2156 LAPACKE_zlarfx_work 5101
4.19.2157 LAPACKE_zlarnv 5101
4.19.2158 LAPACKE_zlarnv_work 5102
4.19.2159 LAPACKE_zlascl 5102
4.19.2160 LAPACKE_zlascl_work 5103
4.19.2161 LAPACKE_zlaset 5103
4.19.2162 LAPACKE_zlaset_work 5104
4.19.2163 LAPACKE_zlassq 5104
4.19.2164 LAPACKE_zlassq_work 5105
4.19.2165 LAPACKE_zlaswp 5105
4.19.2166 LAPACKE_zlaswp_work 5106
4.19.2167 LAPACKE_zlatms 5106
4.19.2168 LAPACKE_zlatms_work 5106
4.19.2169 LAPACKE_zlauum 5107
4.19.2170 LAPACKE_zlauum_work 5107
4.19.2171 LAPACKE_zpbcon 5108
4.19.2172 LAPACKE_zpbcon_work 5108
4.19.2173 LAPACKE_zpbequ 5109
4.19.2174 LAPACKE_zpbequ_work 5109
4.19.2175 LAPACKE_zpbrfs 5110
4.19.2176 LAPACKE_zpbrfs_work 5110
4.19.2177 LAPACKE_zpbstf 5111
4.19.2178 LAPACKE_zpbstf_work 5111
4.19.2179 LAPACKE_zpbsv 5112
4.19.2180 LAPACKE_zpbsv_work 5112
4.19.2181 LAPACKE_zpbsvx 5113
4.19.2182 LAPACKE_zpbsvx_work 5113
4.19.2183 LAPACKE_zpbtrf 5114
4.19.2184 LAPACKE_zpbtrf_work 5114
4.19.2185 LAPACKE_zpbtrs 5115
4.19.2186 LAPACKE_zpbtrs_work 5116
4.19.2187 LAPACKE_zpftrf 5116
4.19.2188 LAPACKE_zpftrf_work 5117
4.19.2189 LAPACKE_zpftri 5117
4.19.2190 LAPACKE_zpftri_work 5118
4.19.2191 LAPACKE_zpftrs 5118
4.19.2192 LAPACKE_zpftrs_work 5119
4.19.2193 LAPACKE_zpocon 5119
4.19.2194 LAPACKE_zpocon_work 5119
4.19.2195 LAPACKE_zpoequ 5120
4.19.2196 LAPACKE_zpoequ_work 5121
4.19.2197 LAPACKE_zpoequb 5121
4.19.2198 LAPACKE_zpoequb_work 5122
4.19.2199 LAPACKE_zporfs 5122
4.19.2200 LAPACKE_zporfs_work 5123
4.19.2201 LAPACKE_zporfsx 5123

4.19.2202 LAPACKE_zporfsx_work 5124
4.19.2203 LAPACKE_zposv 5125
4.19.2204 LAPACKE_zposv_work 5125
4.19.2205 LAPACKE_zposvx 5126
4.19.2206 LAPACKE_zposvx_work 5126
4.19.2207 LAPACKE_zposvxx 5127
4.19.2208 LAPACKE_zposvxx_work 5127
4.19.2209 LAPACKE_zpotrf 5128
4.19.2210 LAPACKE_zpotrf2 5129
4.19.2211 LAPACKE_zpotrf2_work 5129
4.19.2212 LAPACKE_zpotrf_work 5130
4.19.2213 LAPACKE_zpotri 5130
4.19.2214 LAPACKE_zpotri_work 5131
4.19.2215 LAPACKE_zpotrs 5131
4.19.2216 LAPACKE_zpotrs_work 5132
4.19.2217 LAPACKE_zppcon 5132
4.19.2218 LAPACKE_zppcon_work 5133
4.19.2219 LAPACKE_zppequ 5133
4.19.2220 LAPACKE_zppequ_work 5134
4.19.2221 LAPACKE_zpprfs 5134
4.19.2222 LAPACKE_zpprfs_work 5135
4.19.2223 LAPACKE_zppsv 5135
4.19.2224 LAPACKE_zppsv_work 5136
4.19.2225 LAPACKE_zppsvx 5136
4.19.2226 LAPACKE_zppsvx_work 5137
4.19.2227 LAPACKE_zpptrf 5137
4.19.2228 LAPACKE_zpptrf_work 5138
4.19.2229 LAPACKE_zpptri 5138
4.19.2230 LAPACKE_zpptri_work 5139
4.19.2231 LAPACKE_zpptrs 5139
4.19.2232 LAPACKE_zpptrs_work 5140
4.19.2233 LAPACKE_zpstrf 5141
4.19.2234 LAPACKE_zpstrf_work 5141
4.19.2235 LAPACKE_zptcon 5142
4.19.2236 LAPACKE_zptcon_work 5142
4.19.2237 LAPACKE_zpteqr 5143
4.19.2238 LAPACKE_zpteqr_work 5143
4.19.2239 LAPACKE_zptrfs 5144
4.19.2240 LAPACKE_zptrfs_work 5144
4.19.2241 LAPACKE_zptsv 5145
4.19.2242 LAPACKE_zptsv_work 5145
4.19.2243 LAPACKE_zptsvx 5146
4.19.2244 LAPACKE_zptsvx_work 5146
4.19.2245 LAPACKE_zpttrf 5147
4.19.2246 LAPACKE_zpttrf_work 5147
4.19.2247 LAPACKE_zpttrs 5148
4.19.2248 LAPACKE_zpttrs_work 5148
4.19.2249 LAPACKE_zspcon 5149
4.19.2250 LAPACKE_zspcon_work 5149
4.19.2251 LAPACKE_zsprfs 5150
4.19.2252 LAPACKE_zsprfs_work 5150
4.19.2253 LAPACKE_zspsv 5151
4.19.2254 LAPACKE_zspsv_work 5152
4.19.2255 LAPACKE_zspsvx 5152
4.19.2256 LAPACKE_zspsvx_work 5153
4.19.2257 LAPACKE_zsptrf 5153
4.19.2258 LAPACKE_zsptrf_work 5154
4.19.2259 LAPACKE_zsptri 5154

4.19.2260 LAPACKE_zsptri_work 5155
4.19.2261 LAPACKE_zsptrs 5155
4.19.2262 LAPACKE_zsptrs_work 5156
4.19.2263 LAPACKE_zstedc 5156
4.19.2264 LAPACKE_zstedc_work 5157
4.19.2265 LAPACKE_zstegr 5157
4.19.2266 LAPACKE_zstegr_work 5158
4.19.2267 LAPACKE_zstein 5158
4.19.2268 LAPACKE_zstein_work 5159
4.19.2269 LAPACKE_zstemr 5160
4.19.2270 LAPACKE_zstemr_work 5160
4.19.2271 LAPACKE_zsteqr 5161
4.19.2272 LAPACKE_zsteqr_work 5161
4.19.2273 LAPACKE_zsycon 5162
4.19.2274 LAPACKE_zsycon_3 5162
4.19.2275 LAPACKE_zsycon_3_work 5163
4.19.2276 LAPACKE_zsycon_work 5163
4.19.2277 LAPACKE_zsyconv 5164
4.19.2278 LAPACKE_zsyconv_work 5164
4.19.2279 LAPACKE_zsyequb 5165
4.19.2280 LAPACKE_zsyequb_work 5165
4.19.2281 LAPACKE_zsyr 5166
4.19.2282 LAPACKE_zsyr_work 5166
4.19.2283 LAPACKE_zsyrfs 5167
4.19.2284 LAPACKE_zsyrfs_work 5167
4.19.2285 LAPACKE_zsyrfsx 5168
4.19.2286 LAPACKE_zsyrfsx_work 5169
4.19.2287 LAPACKE_zsysv 5169
4.19.2288 LAPACKE_zsysv_aa 5170
4.19.2289 LAPACKE_zsysv_aa_2stage 5170
4.19.2290 LAPACKE_zsysv_aa_work 5171
4.19.2291 LAPACKE_zsysv_rk 5172
4.19.2292 LAPACKE_zsysv_rk_work 5172
4.19.2293 LAPACKE_zsysv_rook 5173
4.19.2294 LAPACKE_zsysv_rook_work 5173
4.19.2295 LAPACKE_zsysv_work 5174
4.19.2296 LAPACKE_zsysvx 5174
4.19.2297 LAPACKE_zsysvx_work 5175
4.19.2298 LAPACKE_zsysvxx 5175
4.19.2299 LAPACKE_zsysvxx_work 5176
4.19.2300 LAPACKE_zsyswapr 5177
4.19.2301 LAPACKE_zsyswapr_work 5177
4.19.2302 LAPACKE_zsytrf 5178
4.19.2303 LAPACKE_zsytrf_aa 5178
4.19.2304 LAPACKE_zsytrf_aa_2stage 5179
4.19.2305 LAPACKE_zsytrf_aa_work 5179
4.19.2306 LAPACKE_zsytrf_rk 5180
4.19.2307 LAPACKE_zsytrf_rk_work 5180
4.19.2308 LAPACKE_zsytrf_rook 5181
4.19.2309 LAPACKE_zsytrf_rook_work 5181
4.19.2310 LAPACKE_zsytrf_work 5182
4.19.2311 LAPACKE_zsytri 5182
4.19.2312 LAPACKE_zsytri2 5183
4.19.2313 LAPACKE_zsytri2_work 5183
4.19.2314 LAPACKE_zsytri2x 5184
4.19.2315 LAPACKE_zsytri2x_work 5184
4.19.2316 LAPACKE_zsytri_3 5185
4.19.2317 LAPACKE_zsytri_3_work 5185

4.19.2318 LAPACKE_zsytri_work 5186
4.19.2319 LAPACKE_zsytrs 5186
4.19.2320 LAPACKE_zsytrs2 5187
4.19.2321 LAPACKE_zsytrs2_work 5187
4.19.2322 LAPACKE_zsytrs_3 5188
4.19.2323 LAPACKE_zsytrs_3_work 5189
4.19.2324 LAPACKE_zsytrs_aa 5189
4.19.2325 LAPACKE_zsytrs_aa_2stage 5190
4.19.2326 LAPACKE_zsytrs_aa_work 5190
4.19.2327 LAPACKE_zsytrs_rook 5191
4.19.2328 LAPACKE_zsytrs_rook_work 5191
4.19.2329 LAPACKE_zsytrs_work 5192
4.19.2330 LAPACKE_ztbcon 5192
4.19.2331 LAPACKE_ztbcon_work 5193
4.19.2332 LAPACKE_ztbrfs 5194
4.19.2333 LAPACKE_ztbrfs_work 5194
4.19.2334 LAPACKE_ztbtrs 5195
4.19.2335 LAPACKE_ztbtrs_work 5195
4.19.2336 LAPACKE_ztfsm 5196
4.19.2337 LAPACKE_ztfsm_work 5196
4.19.2338 LAPACKE_ztftri 5197
4.19.2339 LAPACKE_ztftri_work 5197
4.19.2340 LAPACKE_ztfttp 5198
4.19.2341 LAPACKE_ztfttp_work 5198
4.19.2342 LAPACKE_ztfttr 5199
4.19.2343 LAPACKE_ztfttr_work 5199
4.19.2344 LAPACKE_ztgevc 5200
4.19.2345 LAPACKE_ztgevc_work 5200
4.19.2346 LAPACKE_ztgexc 5201
4.19.2347 LAPACKE_ztgexc_work 5202
4.19.2348 LAPACKE_ztgsen 5202
4.19.2349 LAPACKE_ztgsen_work 5203
4.19.2350 LAPACKE_ztgsja 5203
4.19.2351 LAPACKE_ztgsja_work 5204
4.19.2352 LAPACKE_ztgsna 5205
4.19.2353 LAPACKE_ztgsna_work 5205
4.19.2354 LAPACKE_ztgsyl 5206
4.19.2355 LAPACKE_ztgsyl_work 5206
4.19.2356 LAPACKE_ztpcon 5207
4.19.2357 LAPACKE_ztpcon_work 5208
4.19.2358 LAPACKE_ztpmqrt 5208
4.19.2359 LAPACKE_ztpmqrt_work 5209
4.19.2360 LAPACKE_ztpqrt 5209
4.19.2361 LAPACKE_ztpqrt2 5210
4.19.2362 LAPACKE_ztpqrt2_work 5210
4.19.2363 LAPACKE_ztpqrt_work 5211
4.19.2364 LAPACKE_ztprfb 5211
4.19.2365 LAPACKE_ztprfb_work 5212
4.19.2366 LAPACKE_ztprfs 5213
4.19.2367 LAPACKE_ztprfs_work 5213
4.19.2368 LAPACKE_ztptri 5214
4.19.2369 LAPACKE_ztptri_work 5214
4.19.2370 LAPACKE_ztptrs 5215
4.19.2371 LAPACKE_ztptrs_work 5215
4.19.2372 LAPACKE_ztpttf 5216
4.19.2373 LAPACKE_ztpttf_work 5216
4.19.2374 LAPACKE_ztpttr 5217
4.19.2375 LAPACKE_ztpttr_work 5217

4.19.2376 LAPACKE_ztrcon 5218
4.19.2377 LAPACKE_ztrcon_work 5218
4.19.2378 LAPACKE_ztrevc 5219
4.19.2379 LAPACKE_ztrevc_work 5219
4.19.2380 LAPACKE_ztrexc 5220
4.19.2381 LAPACKE_ztrexc_work 5220
4.19.2382 LAPACKE_ztrrfs 5221
4.19.2383 LAPACKE_ztrrfs_work 5221
4.19.2384 LAPACKE_ztrsen 5222
4.19.2385 LAPACKE_ztrsen_work 5223
4.19.2386 LAPACKE_ztrsna 5223
4.19.2387 LAPACKE_ztrsna_work 5224
4.19.2388 LAPACKE_ztrsyl 5224
4.19.2389 LAPACKE_ztrsyl_work 5225
4.19.2390 LAPACKE_ztrtri 5225
4.19.2391 LAPACKE_ztrtri_work 5226
4.19.2392 LAPACKE_ztrtrs 5226
4.19.2393 LAPACKE_ztrtrs_work 5227
4.19.2394 LAPACKE_ztrttf 5227
4.19.2395 LAPACKE_ztrttf_work 5228
4.19.2396 LAPACKE_ztrttp 5228
4.19.2397 LAPACKE_ztrttp_work 5229
4.19.2398 LAPACKE_ztzrzf 5229
4.19.2399 LAPACKE_ztzrzf_work 5230
4.19.2400 LAPACKE_zunbdb 5230
4.19.2401 LAPACKE_zunbdb_work 5231
4.19.2402 LAPACKE_zuncsd 5232
4.19.2403 LAPACKE_zuncsd2by1 5232
4.19.2404 LAPACKE_zuncsd2by1_work 5233
4.19.2405 LAPACKE_zuncsd_work 5234
4.19.2406 LAPACKE_zungbr 5234
4.19.2407 LAPACKE_zungbr_work 5235
4.19.2408 LAPACKE_zunghr 5235
4.19.2409 LAPACKE_zunghr_work 5236
4.19.2410 LAPACKE_zunglq 5236
4.19.2411 LAPACKE_zunglq_work 5237
4.19.2412 LAPACKE_zungql 5237
4.19.2413 LAPACKE_zungql_work 5238
4.19.2414 LAPACKE_zungqr 5238
4.19.2415 LAPACKE_zungqr_work 5239
4.19.2416 LAPACKE_zungrq 5239
4.19.2417 LAPACKE_zungrq_work 5240
4.19.2418 LAPACKE_zungtr 5240
4.19.2419 LAPACKE_zungtr_work 5241
4.19.2420 LAPACKE_zunmbr 5241
4.19.2421 LAPACKE_zunmbr_work 5242
4.19.2422 LAPACKE_zunmhr 5242
4.19.2423 LAPACKE_zunmhr_work 5243
4.19.2424 LAPACKE_zunmlq 5244
4.19.2425 LAPACKE_zunmlq_work 5244
4.19.2426 LAPACKE_zunmql 5245
4.19.2427 LAPACKE_zunmql_work 5245
4.19.2428 LAPACKE_zunmqr 5246
4.19.2429 LAPACKE_zunmqr_work 5246
4.19.2430 LAPACKE_zunmrq 5247
4.19.2431 LAPACKE_zunmrq_work 5247
4.19.2432 LAPACKE_zunmrz 5248
4.19.2433 LAPACKE_zunmrz_work 5248

4.19.2434 LAPACKE_zunmtr . . . 5249
4.19.2435 LAPACKE_zunmtr_work . . . 5250
4.19.2436 LAPACKE_zupgtr . . . 5250
4.19.2437 LAPACKE_zupgtr_work . . . 5251
4.19.2438 LAPACKE_zupmtr . . . 5251
4.19.2439 LAPACKE_zupmtr_work . . . 5252

5 Fast Fourier Transforms FFTs 5253
5.1 Fast Fourier Transforms FFTs Introduction . . . 5253
5.2 How to choose which FFT routine you need . . . 5253
5.2.1 The basic interface . . . 5254
5.2.2 The advanced interface . . . 5254
5.2.3 The guru interface . . . 5254
5.3 FFT complex to complex functions . . . 5255
5.3.1 fftw_plan_dft . . . 5255
5.3.2 fftw_plan_dft_1d . . . 5257
5.3.3 fftw_plan_dft_2d . . . 5259
5.3.4 fftw_plan_dft_3d . . . 5261
5.3.5 fftw_plan_guru_dft . . . 5263
5.3.6 fftw_plan_guru_split_dft . . . 5266
5.3.7 fftw_plan_guru64_dft . . . 5269
5.3.8 fftw_plan_guru64_split_dft . . . 5271
5.3.9 fftw_plan_many_dft . . . 5274
5.3.10 fftwf_plan_dft . . . 5277
5.3.11 fftwf_plan_dft_1d . . . 5279
5.3.12 fftwf_plan_dft_2d . . . 5281
5.3.13 fftwf_plan_dft_3d . . . 5283
5.3.14 fftwf_plan_guru_dft . . . 5285
5.3.15 fftwf_plan_guru_split_dft . . . 5288
5.3.16 fftwf_plan_guru64_dft . . . 5290
5.3.17 fftwf_plan_guru64_split_dft . . . 5293
5.3.18 fftwf_plan_many_dft . . . 5296
5.3.19 fftwh_plan_dft . . . 5299
5.3.20 fftwh_plan_dft_1d . . . 5301
5.3.21 fftwh_plan_dft_2d . . . 5302
5.3.22 fftwh_plan_dft_3d . . . 5304
5.3.23 fftwh_plan_guru_dft . . . 5306
5.3.24 fftwh_plan_guru_split_dft . . . 5308
5.3.25 fftwh_plan_guru64_dft . . . 5310
5.3.26 fftwh_plan_guru64_split_dft . . . 5313
5.3.27 fftwh_plan_many_dft . . . 5316
5.4 FFT real to complex functions . . . 5318
5.4.1 fftw_plan_dft_r2c . . . 5318
5.4.2 fftw_plan_dft_r2c_1d . . . 5320
5.4.3 fftw_plan_dft_r2c_2d . . . 5322
5.4.4 fftw_plan_dft_r2c_3d . . . 5323
5.4.5 fftw_plan_guru_dft_r2c . . . 5325
5.4.6 fftw_plan_guru_split_dft_r2c . . . 5328
5.4.7 fftw_plan_guru64_dft_r2c . . . 5330
5.4.8 fftw_plan_guru64_split_dft_r2c . . . 5333
5.4.9 fftw_plan_many_dft_r2c . . . 5335
5.4.10 fftwf_plan_dft_r2c . . . 5338
5.4.11 fftwf_plan_dft_r2c_1d . . . 5340
5.4.12 fftwf_plan_dft_r2c_2d . . . 5342
5.4.13 fftwf_plan_dft_r2c_3d . . . 5343
5.4.14 fftwf_plan_guru_dft_r2c . . . 5345
5.4.15 fftwf_plan_guru_split_dft_r2c . . . 5348
5.4.16 fftwf_plan_guru64_dft_r2c . . . 5350

5.4.17 fftwf_plan_guru64_split_dft_r2c . . . 5353
5.4.18 fftwf_plan_many_dft_r2c . . . 5355
5.4.19 fftwh_plan_dft_r2c . . . 5358
5.4.20 fftwh_plan_dft_r2c_1d . . . 5360
5.4.21 fftwh_plan_dft_r2c_2d . . . 5361
5.4.22 fftwh_plan_dft_r2c_3d . . . 5363
5.4.23 fftwh_plan_guru_dft_r2c . . . 5364
5.4.24 fftwh_plan_guru_split_dft_r2c . . . 5367
5.4.25 fftwh_plan_guru64_dft_r2c . . . 5369
5.4.26 fftwh_plan_guru64_split_dft_r2c . . . 5371
5.4.27 fftwh_plan_many_dft_r2c . . . 5374
5.5 FFT complex to real functions . . . 5376
5.5.1 fftw_plan_dft_c2r . . . 5376
5.5.2 fftw_plan_dft_c2r_1d . . . 5378
5.5.3 fftw_plan_dft_c2r_2d . . . 5379
5.5.4 fftw_plan_dft_c2r_3d . . . 5381
5.5.5 fftw_plan_guru_dft_c2r . . . 5383
5.5.6 fftw_plan_guru_split_dft_c2r . . . 5386
5.5.7 fftw_plan_guru64_dft_c2r . . . 5388
5.5.8 fftw_plan_guru64_split_dft_c2r . . . 5391
5.5.9 fftw_plan_many_dft_c2r . . . 5393
5.5.10 fftwf_plan_dft_c2r . . . 5396
5.5.11 fftwf_plan_dft_c2r_1d . . . 5398
5.5.12 fftwf_plan_dft_c2r_2d . . . 5399
5.5.13 fftwf_plan_dft_c2r_3d . . . 5401
5.5.14 fftwf_plan_guru_dft_c2r . . . 5403
5.5.15 fftwf_plan_guru_split_dft_c2r . . . 5406
5.5.16 fftwf_plan_guru64_dft_c2r . . . 5408
5.5.17 fftwf_plan_guru64_split_dft_c2r . . . 5411
5.5.18 fftwf_plan_many_dft_c2r . . . 5413
5.5.19 fftwh_plan_dft_c2r . . . 5416
5.5.20 fftwh_plan_dft_c2r_1d . . . 5418
5.5.21 fftwh_plan_dft_c2r_2d . . . 5419
5.5.22 fftwh_plan_dft_c2r_3d . . . 5421
5.5.23 fftwh_plan_guru_dft_c2r . . . 5422
5.5.24 fftwh_plan_guru_split_dft_c2r . . . 5424
5.5.25 fftwh_plan_guru64_dft_c2r . . . 5427
5.5.26 fftwh_plan_guru64_split_dft_c2r . . . 5429
5.5.27 fftwh_plan_many_dft_c2r . . . 5431
5.6 FFT real to real functions . . . 5434
5.6.1 fftw_plan_guru_r2r . . . 5434
5.6.2 fftw_plan_guru64_dft_r2r . . . 5436
5.6.3 fftw_plan_many_r2r . . . 5439
5.6.4 fftw_plan_r2r . . . 5442
5.6.5 fftw_plan_r2r_1d . . . 5444
5.6.6 fftw_plan_r2r_2d . . . 5446
5.6.7 fftw_plan_r2r_3d . . . 5448
5.6.8 fftwf_plan_guru_r2r . . . 5450
5.6.9 fftwf_plan_guru64_dft . . . 5452
5.6.10 fftwf_plan_many_r2r . . . 5455
5.6.11 fftwf_plan_r2r . . . 5458
5.6.12 fftwf_plan_r2r_1d . . . 5460
5.6.13 fftwf_plan_r2r_2d . . . 5462
5.6.14 fftwf_plan_r2r_3d . . . 5464
5.6.15 fftwh_plan_guru_r2r . . . 5466
5.6.16 fftwh_plan_guru64_dft . . . 5468
5.6.17 fftwh_plan_many_r2r . . . 5471
5.6.18 fftwh_plan_r2r . . . 5473

5.6.19 fftwh_plan_r2r_1d . . . 5475
5.6.20 fftwh_plan_r2r_2d . . . 5476
5.6.21 fftwh_plan_r2r_3d . . . 5478
5.7 FFT executors functions . . . 5480
5.7.1 fftw_execute . . . 5480
5.7.2 fftw_execute_dft . . . 5480
5.7.3 fftw_execute_dft_c2r . . . 5481
5.7.4 fftw_execute_dft_r2c . . . 5482
5.7.5 fftw_execute_r2r . . . 5483
5.7.6 fftwf_execute . . . 5484
5.7.7 fftwf_execute_dft . . . 5484
5.7.8 fftwf_execute_dft_c2r . . . 5485
5.7.9 fftwf_execute_dft_r2c . . . 5486
5.7.10 fftwf_execute_r2r . . . 5486
5.7.11 fftwh_execute . . . 5487
5.7.12 fftwh_execute_dft . . . 5488
5.7.13 fftwh_execute_dft_c2r . . . 5488
5.7.14 fftwh_execute_dft_r2c . . . 5489
5.7.15 fftwh_execute_r2r . . . 5489
5.8 FFT memory functions . . . 5490
5.8.1 fftw_alignment_of . . . 5490
5.8.2 fftw_alloc_complex . . . 5491
5.8.3 fftw_alloc_real . . . 5491
5.8.4 fftw_cleanup . . . 5492
5.8.5 fftw_destroy_plan . . . 5492
5.8.6 fftw_free . . . 5493
5.8.7 fftw_malloc . . . 5493
5.8.8 fftwf_alignment_of . . . 5494
5.8.9 fftwf_alloc_complex . . . 5495
5.8.10 fftwf_alloc_real . . . 5495
5.8.11 fftwf_cleanup . . . 5496
5.8.12 fftwf_destroy_plan . . . 5496
5.8.13 fftwf_free . . . 5497
5.8.14 fftwf_malloc . . . 5497
5.8.15 fftwh_alignment_of . . . 5498
5.8.16 fftwh_alloc_complex . . . 5498
5.8.17 fftwh_alloc_real . . . 5499
5.8.18 fftwh_cleanup . . . 5499
5.8.19 fftwh_destroy_plan . . . 5499
5.8.20 fftwh_free . . . 5500
5.8.21 fftwh_malloc . . . 5500
5.9 FFT wisdom functions . . . 5501
5.9.1 fftw_export_wisdom . . . 5501
5.9.2 fftw_export_wisdom_to_file . . . 5501
5.9.3 _armpl_fftw_export_wisdom_to_filname . . . 5502
5.9.4 fftw_export_wisdom_to_string . . . 5503
5.9.5 fftw_forget_wisdom . . . 5503
5.9.6 fftw_import_system_wisdom . . . 5504
5.9.7 fftw_import_wisdom . . . 5504
5.9.8 fftw_import_wisdom_from_file . . . 5505
5.9.9 fftw_import_wisdom_from_filename . . . 5506
5.9.10 fftw_import_wisdom_from_string . . . 5506
5.9.11 fftwf_export_wisdom . . . 5507
5.9.12 fftwf_export_wisdom_to_file . . . 5508
5.9.13 _armpl_fftwf_export_wisdom_to_filname . . . 5508
5.9.14 fftwf_export_wisdom_to_string . . . 5509
5.9.15 fftwf_forget_wisdom . . . 5509
5.9.16 fftwf_import_system_wisdom . . . 5510

5.9.17 fftwf_import_wisdom . . . 5510
5.9.18 fftwf_import_wisdom_from_file . . . 5511
5.9.19 fftwf_import_wisdom_from_filename . . . 5512
5.9.20 fftwf_import_wisdom_from_string . . . 5512
5.9.21 fftwh_export_wisdom . . . 5513
5.9.22 fftwh_export_wisdom_to_file . . . 5514
5.9.23 _armpl_fftwh_export_wisdom_to_filname . . . 5514
5.9.24 fftwh_export_wisdom_to_string . . . 5514
5.9.25 fftwh_forget_wisdom . . . 5515
5.9.26 fftwh_import_system_wisdom . . . 5515
5.9.27 fftwh_import_wisdom . . . 5516
5.9.28 fftwh_import_wisdom_from_file . . . 5516
5.9.29 fftwh_import_wisdom_from_filename . . . 5517
5.9.30 fftwh_import_wisdom_from_string . . . 5517
5.10 FFT threading functions . . . 5518
5.10.1 fftw_cleanup_threads . . . 5518
5.10.2 fftw_init_threads . . . 5518
5.10.3 fftw_make_planner_thread_safe . . . 5519
5.10.4 fftw_plan_with_nthreads . . . 5519
5.10.5 fftwf_cleanup_threads . . . 5520
5.10.6 fftwf_init_threads . . . 5520
5.10.7 fftwf_make_planner_thread_safe . . . 5521
5.10.8 fftwf_plan_with_nthreads . . . 5521
5.10.9 fftwh_cleanup_threads . . . 5522
5.10.10 fftwh_init_threads . . . 5522
5.10.11 fftwh_make_planner_thread_safe . . . 5522
5.10.12 fftwh_plan_with_nthreads . . . 5523
5.11 FFT utilities functions . . . 5523
5.11.1 fftw_cost . . . 5523
5.11.2 fftw_estimate_cost . . . 5524
5.11.3 fftw_flops . . . 5525
5.11.4 fftw_fprint_plan . . . 5525
5.11.5 fftw_print_plan . . . 5526
5.11.6 fftw_set_timelimit . . . 5527
5.11.7 fftwf_cost . . . 5527
5.11.8 fftwf_estimate_cost . . . 5528
5.11.9 fftwf_flops . . . 5529
5.11.10 fftwf_fprint_plan . . . 5529
5.11.11 fftwf_print_plan . . . 5530
5.11.12 fftwf_set_timelimit . . . 5531
5.11.13 fftwh_cost . . . 5531
5.11.14 fftwh_estimate_cost . . . 5532
5.11.15 fftwh_flops . . . 5532
5.11.16 fftwh_fprint_plan . . . 5533
5.11.17 fftwh_print_plan . . . 5533
5.11.18 fftwh_set_timelimit . . . 5534

6 Sparse Linear Algebra 5535
6.1 Sparse Linear Algebra Introduction . . . 5535
6.2 Example of SpMV usage . . . 5535
6.3 Sparse Linear Algebra Functions . . . 5537
6.3.1 armpl_spmat_create_coo_c . . . 5537
6.3.2 armpl_spmat_create_coo_d . . . 5539
6.3.3 armpl_spmat_create_coo_s . . . 5541
6.3.4 armpl_spmat_create_coo_z . . . 5543
6.3.5 armpl_spmat_create_csc_c . . . 5545
6.3.6 armpl_spmat_create_csc_d . . . 5546
6.3.7 armpl_spmat_create_csc_s . . . 5548

6.3.8 armpl_spmat_create_csc_z 5550
6.3.9 armpl_spmat_create_csr_c 5552
6.3.10 armpl_spmat_create_csr_d 5554
6.3.11 armpl_spmat_create_csr_s 5555
6.3.12 armpl_spmat_create_csr_z 5557
6.3.13 armpl_spmat_update_c 5559
6.3.14 armpl_spmat_update_d 5561
6.3.15 armpl_spmat_update_s 5562
6.3.16 armpl_spmat_update_z 5564
6.3.17 armpl_spmat_hint 5565
6.3.18 armpl_spmv_optimize 5567
6.3.19 armpl_spmv_exec_c 5568
6.3.20 armpl_spmv_exec_d 5570
6.3.21 armpl_spmv_exec_s 5572
6.3.22 armpl_spmv_exec_z 5573
6.3.23 armpl_spmat_destroy 5575

CHAPTER

ONE

INTRODUCTION

1.1 Arm Performance Libraries

The Arm Performance Libraries are a set of numerical routines tuned specifically for Arm processors. The routines, which are available using both the Fortran and C interfaces, include:

- BLAS: Basic Linear Algebra Subprograms (including XBLAS, the extended precision BLAS).
- LAPACK: A comprehensive package of higher-level linear algebra routines.
- FFT: A set of Fast Fourier Transform routines for real and complex data.
- Sparse: A selection of basic linear algebra routines for working with sparse data.

The BLAS and LAPACK routines provide a portable, standard set of interfaces for common numerical linear algebra operations. Full documentation for BLAS and LAPACK are available online. This manual provides:

- Brief descriptions of BLAS and LAPACK.
- Links to the documentation for BLAS and LAPACK. See *BLAS Basic Linear Algebra Subprograms*, and *LAPACK Linear Algebra Package*.
- Links to other materials.

The FFT is an implementation of the Discrete Fourier Transform (DFT). FFT uses symmetries in the definition to reduce the number of operations that are required from O(n*n) to O(n*log n). FFTs in Arm Performance Libraries are provided using the same interface as the popular Open Source package FFTW3. See *Fast Fourier Transforms FFTs Introduction*.

Sparse routines are provided by using an API that separates the description of a sparse problem from its execution. The basic flow is to first create a handle to a sparse matrix by providing the matrix in a common format (for example Compressed Sparse Rows). Next, the handle is used to identify the matrix in order to describe its characteristics and perform optimizations. Finally, the handle is passed to an execution step. This workflow gives the library the opportunity to create new data structures that are more suited to solving the problem on the hardware.

General Information *General Information* provides the following details:

- Linking a user program to Arm Performance Libraries.
- Fortran and C interfaces to Arm Performance Libraries routines.
- Getting the Arm Performance Libraries version and build information.
- Accessing the Arm Performance Libraries documentation.

1.2 References

- [1] C.L. Lawson, R.J. Hanson, D. Kincaid, and F.T. Krogh, *Basic linear algebra subprograms for Fortran usage*, ACM Trans. Maths. Soft., 5 (1979), pp. 308–323.
- [2] J.J. Dongarra, J. Du Croz, S. Hammarling, and R.J. Hanson, *An extended set of FORTRAN basic linear algebra subroutines*, ACM Trans. Math. Soft., 14 (1988), pp. 1–17.

- [3] J.J. Dongarra, J. Du Croz, I.S. Duff, and S. Hammarling, *A set of level 3 basic linear algebra subprograms*, ACM Trans. Math. Soft., 16 (1990), pp. 1–17.
- [4] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK User's Guide*, SIAM, Philidelphia, (1999).
- [5] IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985)

CHAPTER

TWO

GENERAL INFORMATION

2.1 Accessing the Library

Arm Performance Libraries is compatible with two compilers: GNU GCC (gcc/gfortran) and the Arm C/C++/Fortran Compiler (armclang/armflang) that is packaged with the Arm Allinea Studio. Separate binary distributions of the library are supplied for these two compilers. To manage access to the correct library for your choice of compiler, Arm recommends using the environment modules provided. The environment modules set the `ARMPL_DIR` environment variable appropriately.

Note: If you are not using the environment modules, you can assume that `${ARMPL_DIR}` is set to `/opt/arm/armpl-..._CPU_DISTRO`.

Use the following command to compile the program driver.f90 using `armflang` and link the program to Arm Performance Libraries:

```
armflang driver.f90 -L${ARMPL_DIR}/lib -larmpl_lp64
```

Use the following command to compile the program driver.f90 using `gfortran` and link the program to Arm Performance Libraries:

```
gfortran driver.f90 -L${ARMPL_DIR}/lib -larmpl_lp64
```

The Arm Performance Libraries are supplied in both static and shareable versions, `libarmpl_lp64.a` and `libarmpl_lp64.so`. By default, the commands given above link to the shareable version of the library, `libarmpl_lp64.so`, if that version exists in the specified directory.

To force linking with the static library, use the compiler flag `-static`, for example:

```
armflang driver.f90 -L${ARMPL_DIR}/lib -static -larmpl_lp64 -lrt
```

```
gfortran driver.f90 -L${ARMPL_DIR}/lib -static -larmpl_lp64 -lrt
```

Alternatively, insert the name of the static library in the command line, for example:

```
armflang driver.f90 ${ARMPL_DIR}/lib/libarmpl_lp64.a -lrt
```

```
gfortran driver.f90 ${ARMPL_DIR}/lib/libarmpl_lp64.a -lrt
```

Note: When using the supplied environment modules, check to see that the `LD_LIBRARY_PATH` is automatically updated; if you do not use the supplied environment variable, you must set `LD_LIBRARY_PATH` yourself when linking to the shareable library.

To link against a multi-threaded version of Arm Performance Libraries, use the following commands, as appropriate:

```
armflang -fopenmp driver.f90 -L${ARMPL_DIR}/lib -larmpl_lp64_mp
```

```
gfortran -fopenmp driver.f90 -L${ARMPL_DIR}/lib -larmpl_lp64_mp
```

Note: The library names now include the suffix _mp.

To compile and link a C program with Arm Performance Libraries, use the following commands, as appropriate:

```
armclang -I${ARMPL_DIR}/include driver.c \
         -L${ARMPL_DIR}/lib -larmpl_lp64 -larmflang
```

```
gcc -I${ARMPL_DIR}/include driver.c \
     -L${ARMPL_DIR}/lib -larmpl_lp64 -lgfortran
```

The switch "-I${ARMPL_DIR}/include" tells the compiler to search the specified directory for the Arm Performance Libraries C header file armpl.h, which must be included by driver.c.

Note: When you link the program, you must add the Fortran compiler run-time library or libraries (-lgfortran or -lflang -lflangrti). If you are using the Arm Compiler, you can apply the -armpl flag to load the optimum version of Arm Performance Libraries for your target architecture and implementation. For more information, see Arm Performance Libraries Library Selection.

2.2 Arm Performance Libraries Fortran and C interfaces

Most routines in Arm Performance Libraries come with both Fortran and C interfaces. The Fortran interfaces typically follow the relevant standard, such as LAPACK and BLAS. In C code that uses Arm Performance Libraries routines, ensure that you include the header file armpl.h, which contains function prototypes for all Arm Performance Libraries C interfaces. The header file also contains C prototypes for Fortran interfaces so that you can call the Fortran interfaces from C if you need to do so.

For more details on the C interfaces, see *BLAS Basic Linear Algebra Subprograms*, *LAPACK Linear Algebra Package*, and *Fast Fourier Transforms FFTs Introduction*.

Arm Performance Libraries also includes Fortran interface block modules for user-callable routines. These modules define the type and arguments of each Arm Performance Libraries routine. Their purpose is to allow the Fortran compiler to check that Arm Performance Libraries routines are called correctly. The interface blocks allow the compiler to check that:

- Subroutines are called correctly.
- Functions are declared with the correct type.
- The correct number of arguments are passed.
- All arguments match in type and structure.

The interface block modules are not essential to calling Arm Performance Libraries from Fortran programs. However, the supplied example programs use the interface block modules and Arm recommends that you also use them in order to avoid common mistakes.

The interfaces to Arm Performance Libraries routines are declared in files armpl_blas.f90 and armpl_lapack.f90, and they are also combined into one module named armpl_library. You can find these modules in the Arm Performance Libraries **include** directory. For help on how to use these modules, see the Fortran programs in the Arm Performance Libraries examples directory.

2.3 Arm Performance Libraries variant using 64-bit integer (INTEGER*8) arguments

You can use the `INTEGER*8` version of the Arm Performance Libraries with Fortran programs that use `INTEGER*8` variables, and C programs that use 8-byte long variables.

The `INTEGER*8` versions of the libraries include the string "`_ilp64`" as part of the library name, instead of "`_lp64`" which is used in the name of 32-bit integer versions of the libraries. You can also find additional versions of the module files that are compiled to use `INTEGER*8` in the `include_ilp64` directory.

```
gfortran driver.f90 -L${ARMPL_DR}/lib -larmpl_ilp64 -I${ARMPL_DIR}/include_ilp64
```

```
armflang driver.f90 -L${ARMPL_DIR}/lib -larmpl_ilp64 -I${ARMPL_DIR}/include_ilp64
```

For these Arm Performance Libraries variants, ensure that the documentation which mentions Fortran `INTEGER` argument types refers to them as `INTEGER*8`, and refers to C `int` argument types as `long`.

You must ensure that you link to the ilp64 variant if you have INTEGER*8 variables in your code, and not to other variants. Unexpected program crashes are likely to occur if you link to the wrong version.

2.4 Library Version and Build Information

This document is applicable to version 19.2 of Arm Performance Libraries. The utility routine `armplversion()` can be called to get the major, minor, and patch version numbers of the installed Arm Performance Libraries.

The utility routine `armplinfo()` can be called to get information on the compiler that is used to build Arm Performance Libraries, the version of the compiler, and the options that are used for building the Library. This subroutine takes no arguments and prints the information to the current standard output.

2.4.1 Fortran specifications

SUBROUTINE: **ARMPLVERSION** *(MAJOR, MINOR, PATCH, BUILD)*

INTEGER: **MAJOR**, *MINOR, PATCH, BUILD*

SUBROUTINE: **ARMPLINFO** *()*

2.4.2 C specifications

function: *void* **armplversion** *(int *major, int *minor, int *patch, int *build)*;

function: *void* **armplinfo** *(void)*;

2.5 Example programs calling Arm Performance Libraries

The `examples` subdirectory in the Arm Performance Libraries installation directory contains example programs that show how to call Arm Performance Libraries. It also contains a `GNUmakefile` to build and run the example programs. The directory also includes examples of how to call Fortran and C interfaces. You can use these programs to test an Arm Performance Libraries installation.

You might need to modify some variables in the GNUmakefile before using it, depending on where your copy of the Arm Performance Libraries is installed, and depending on which compiler and flags you want to use.

If you need more example programs showing how to call LAPACK routines from Fortran,see the NAG double precision LAPACK driver routines web page. These examples work when linked with Arm Performance Libraries.

Note: In addition to the example programs, you must also download and compile a small amount of utility code used by the programs. See the web page for detailed instructions.

CHAPTER

THREE

BLAS BASIC LINEAR ALGEBRA SUBPROGRAMS

3.1 Overview

The BLAS are a set of well defined basic linear algebra operations ([1], [2], [3]). These operations are subdivided into three groups:

- Level 1: operations acting on vectors only (such as, dot product)
- Level 2: matrix-vector operations (such as, matrix-vector multiplication)
- Level 3: matrix-matrix operations (such as, matrix-matrix multiplication)

Efficient machine-specific implementations of the BLAS are available for many modern high-performance computers. The implementation of higher level linear algebra algorithms on these systems depends critically on the use of the BLAS as building blocks.

For any information relating to the standard BLAS Fortran interfaces please refer to the BLAS FAQ:

http://www.netlib.org/blas/faq.html

For C users, Arm Performance Libraries includes CBLAS, which are C interfaces to the Fortran BLAS. Names of the CBLAS routines are identical to the Fortran routines except that they are prefixed by the string `cblas_`. For example, the Fortran BLAS routine `dgemm` becomes `cblas_dgemm` in C. Other differences between Fortran BLAS and CBLAS interfaces:

- Scalar input arguments are passed by value in CBLAS interfaces. Fortran interfaces pass all arguments by reference (except for character string *length* arguments that are normally hidden from Fortran programmers).
- Unlike Fortran, C has no native *complex* data type. Arm Performance Libraries C routines, which operate on complex data, use the types `armpl_singlecomplex_t` and `armpl_doublecomplex_t` defined in `armpl.h` for single and double precision computations, respectively.
- As with Fortran, all CBLAS array arguments must be stored in contiguous memory. However, all Fortran BLAS routines which take one or more matrices (2D arrays) as arguments have an extra parameter in the CBLAS interface, which appears before all other parameters. This parameter is of the enum type `CBLAS_ORDER`, and it can take one of the values `CblasRowMajor` or `CblasColMajor`.

  The value `CblasColMajor` indicates that elements within any column of each array are contiguous in memory while elements within array rows are offset by a constant stride. This is the usual Fortran storage order. Such strides are given by "leading dimension" arguments (such as `LDA`) in the Fortran and C interfaces.

  The value `CblasRowMajor` indicates that elements within any row of each 2D array are contiguous in memory while elements within array columns are offset by a constant stride such as `LDA`.
- Fortran's `character` type arguments are converted into C enumerated types, as shown in the table below:

Table 1: Fortran and CBLAS interfaces

Fortran Interface		CBLAS Interface	
Character Argument	Value	enum type argument	Value
SIDE	'L' 'R'	CBLAS_SIDE	CblasLeft CblasRight
UPLO	'L' 'U'	CBLAS_UPLO	CblasLower CblasUpper
DIAG	'N' 'U'	CBLAS_DIAG	CblasNonUnit CblasUnit
TRANSPOSE	'T' 'C'	CBLAS_TRANSPOSE	CblasNoTrans CblasRTrans CblasConjTrans

See the `armpl.h` header file to find prototypes for all CBLAS functions.

Note: If you include `armpl.h`, the functions can also be called from a C++ program.

3.2 BLAS level 1

3.2.1 caxpy

`caxpy` performs the following vector update:

$$y \leftarrow \alpha x + y$$

Syntax

Fortran specification:

```
use armpl_library

subroutine caxpy(N, CA, CX, INCX, CY, INCY)
```

C specification:

```
#include "armpl.h"

void caxpy_(const armpl_int_t *n, const armpl_singlecomplex_t *alpha,
            const armpl_singlecomplex_t *x, const armpl_int_t *incx,
            armpl_singlecomplex_t *y, const armpl_int_t *incy);
```

Parameters

N Input parameter.

`N` is `INTEGER`

`N` is the number of elements in the input vectors `CX` and `CY`.

CA Input parameter.

`CA` is `COMPLEX`

`CA` specifies the scalar α.

CX Input parameter.

CX is COMPLEX

CX is the array x, dimension (1 + (N - 1)*abs(INCX)).

INCX Input parameter.

INCX is INTEGER

INCX is the storage spacing between elements of CX.

CY Input and output parameter.

CY is COMPLEX

CY is the array y, dimension (1 + (N - 1)*abs(INCY)).

INCY Input parameter.

INCY is INTEGER

INCY is the storage spacing between elements of CY.

Related Information

For this routine in other precisions, please see *daxpy*, *saxpy* and *zaxpy*. It also exists with a native C interface as *cblas_caxpy*.

3.2.2 ccopy

ccopy copies a vector x to a vector y.

Syntax

Fortran specification:

```
use armpl_library

subroutine ccopy(N, CX, INCX, CY, INCY)
```

C specification:

```
#include "armpl.h"

void ccopy_(const armpl_int_t *n, const armpl_singlecomplex_t *x,
            const armpl_int_t *incx, armpl_singlecomplex_t *y,
            const armpl_int_t *incy);
```

Parameters

N Input parameter.

N is INTEGER

N is the number of elements in the input vectors.

CX Input parameter.

CX is COMPLEX

CX is the array x, dimension (1 + (N - 1)*abs(INCX)).

INCX Input parameter.

INCX is INTEGER

INCX is the storage spacing between elements of CX.

CY Output parameter.

CY is COMPLEX

CY is the array y, dimension (1 + (N - 1)*abs(INCY)).

INCY Input parameter.

INCY is INTEGER

INCY is the storage spacing between elements of CY.

Related Information

For this routine in other precisions, please see *dcopy*, *scopy* and *zcopy*. It also exists with a native C interface as *cblas_ccopy*.

3.2.3 cdotc

cdotc forms the dot product of two complex vectors.

$$CDOTC = X^H \cdot Y$$

Syntax

Fortran specification:

```
use armpl_library

complex function cdotc(N, CX, INCX, CY, INCY)
```

C specification:

```
#include "armpl.h"

armpl_singlecomplex_t cdotc_(const armpl_int_t *n,
                             const armpl_singlecomplex_t *x,
                             const armpl_int_t *incx,
                             const armpl_singlecomplex_t *y,
                             const armpl_int_t *incy);
```

Returns

This returns the result of the dot product, CDOTC.

Parameters

N Input parameter.

N is INTEGER

number of elements in input vector(s)

CX Input parameter.

CX is REAL

CX is an array, dimension (1 + (N - 1)*abs(INCX))

INCX Input parameter.

INCX is INTEGER

storage spacing between elements of CX

CY Input parameter.

CY is REAL

CY is an array, dimension (1 + (N - 1)*abs(INCY))

INCY Input parameter.

INCY is INTEGER

storage spacing between elements of CY

Related Information

For this routine in other precisions, please see *zdotc*.

3.2.4 cdotu

cdotu forms the dot product of two complex vectors.

$$CDOTU = X^T \cdot Y$$

Syntax

Fortran specification:

```
use armpl_library

complex function cdotu(N, CX, INCX, CY, INCY)
```

C specification:

```
#include "armpl.h"

armpl_singlecomplex_t cdotu_(const armpl_int_t *n,
                             const armpl_singlecomplex_t *x,
                             const armpl_int_t *incx,
                             const armpl_singlecomplex_t *y,
                             const armpl_int_t *incy);
```

Returns

This returns the result of the dot product, CDOTU.

Parameters

N Input parameter.

N is INTEGER

number of elements in input vector(s)

CX Input parameter.

CX is REAL

CX is an array, dimension (1 + (N - 1)*abs(INCX))

INCX Input parameter.

INCX is INTEGER

storage spacing between elements of CX

CY Input parameter.

CY is REAL

CY is an array, dimension (1 + (N - 1)*abs(INCY))

INCY Input parameter.

INCY is INTEGER

storage spacing between elements of CY

Related Information

For this routine in other precisions, please see *zdotu*.

3.2.5 crot

crot applies a plane rotation, where the cos (C) is real and the sin (S) is complex, and the vectors CX and CY are complex.

Syntax

Fortran specification:

```
use armpl_library

subroutine crot(N, CX, INCX, CY, INCY, C, S)
```

C specification:

```
#include "armpl.h"

void crot_(const armpl_int_t *n, armpl_singlecomplex_t *cx,
           const armpl_int_t *incx, armpl_singlecomplex_t *cy,
           const armpl_int_t *incy, const float *c,
           const armpl_singlecomplex_t *s);
```

Parameters

N Input parameter.

N is INTEGER

The number of elements in the vectors CX and CY.

CX Input and output parameter.

CX is COMPLEX

CX is an array, dimension (N). On input, the vector X. On output, CX is overwritten with C*X + S*Y.

INCX Input parameter.

INCX is INTEGER

The increment between successive values of CY. INCX <> 0.

CY Input and output parameter.

CY is COMPLEX

CY is an array, dimension (N). On input, the vector Y. On output, CY is overwritten with -CONJG(S)*X + C*Y.

INCY Input parameter.

INCY is INTEGER

The increment between successive values of CY. INCX <> 0.

C Input parameter.

C is REAL

S Input parameter.

S is COMPLEX

C and S define a rotation [C S] [-conjg(S) C] where C* C + S*CONJG(S) = 1.0.

Related Information

For this routine in other precisions, please see *drot*, *srot* and *zrot*.

3.2.6 crotg

crotg determines a complex Givens rotation.

Syntax

Fortran specification:

```
use armpl_library

subroutine crotg(CA, CB, C, S)
```

C specification:

```
#include "armpl.h"

void crotg_(armpl_singlecomplex_t *ca, const armpl_singlecomplex_t *cb,
            float *c, armpl_singlecomplex_t *s);
```

Parameters

CA Input parameter.

CA is COMPLEX

CB Input parameter.

CB is COMPLEX

C Output parameter.

C is REAL

S Output parameter.

S is COMPLEX

Related Information

For this routine in other precisions, please see *drotg*, *srotg* and *zrotg*.

3.2.7 cscal

```
CSCAL scales a vector by a constant.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine cscal(N, CA, CX, INCX)
```

C specification:

```
#include "armpl.h"

void cscal_(const armpl_int_t *n, const armpl_singlecomplex_t *alpha,
            armpl_singlecomplex_t *x, const armpl_int_t *incx);
```

Parameters

N Input parameter.

N is INTEGER

number of elements in input vector(s)

CA Input parameter.

CA is COMPLEX

On entry, CA specifies the scalar alpha.

CX Input and output parameter.

CX is COMPLEX

CX is an array, dimension (1 + (N - 1)*abs(INCX))

INCX Input parameter.

INCX is INTEGER

storage spacing between elements of CX

Related Information

For this routine in other precisions, please see *dscal*, *sscal* and *zscal*. It also exists with a native C interface as *cblas_cscal*.

3.2.8 csrot

csrot applies a plane rotation, where the cos and sin (c and s) are real and the vectors cx and cy are complex. jack dongarra, linpack, 3/11/78.

Syntax

Fortran specification:

```
use armpl_library

subroutine csrot(N, CX, INCX, CY, INCY, C, S)
```

C specification:

```
#include "armpl.h"

void csrot_(const armpl_int_t *n, armpl_singlecomplex_t *x,
            const armpl_int_t *incx, armpl_singlecomplex_t *y,
            const armpl_int_t *incy, const float *c, const float *s);
```

Parameters

N Input parameter.

N is INTEGER

On entry, N specifies the order of the vectors cx and cy. N must be at least zero.

CX Input and output parameter.

CX is COMPLEX

CX is an array, dimension at least. (1 + (N - 1)*abs(INCX)). Before entry, the incremented array CX must contain the n element vector cx. On exit, CX is overwritten by the updated vector cx.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of CX. INCX must not be zero.

CY Input and output parameter.

CY is COMPLEX

CY is an array, dimension at least. (1 + (N - 1)*abs(INCY)). Before entry, the incremented array CY must contain the n element vector cy. On exit, CY is overwritten by the updated vector cy.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of CY. INCY must not be zero.

C Input parameter.

C is REAL

On entry, C specifies the cosine, cos.

S Input parameter.

S is REAL

On entry, S specifies the sine, sin.

Related Information

3.2.9 csscal

CSSCAL scales a complex vector by a real constant.

Syntax

Fortran specification:

```
use armpl_library

subroutine csscal(N, SA, CX, INCX)
```

C specification:

```
#include "armpl.h"

void csscal_(const armpl_int_t *n, const float *alpha,
             armpl_singlecomplex_t *x, const armpl_int_t *incx);
```

Parameters

N Input parameter.

N is INTEGER

number of elements in input vector(s)

SA Input parameter.

SA is REAL

On entry, SA specifies the scalar alpha.

CX Input and output parameter.

CX is COMPLEX

CX is an array, dimension (1 + (N - 1)*abs(INCX))

INCX Input parameter.

INCX is INTEGER

storage spacing between elements of CX

Related Information

It also exists with a native C interface as *cblas_csscal*.

3.2.10 cswap

CSWAP interchanges two vectors.

Syntax

Fortran specification:

```
use armpl_library

subroutine cswap(N, CX, INCX, CY, INCY)
```

C specification:

```
#include "armpl.h"

void cswap_(const armpl_int_t *n, armpl_singlecomplex_t *x,
            const armpl_int_t *incx, armpl_singlecomplex_t *y,
            const armpl_int_t *incy);
```

Parameters

N Input parameter.

N is INTEGER

number of elements in input vector(s)

CX Input and output parameter.

CX is COMPLEX

CX is an array, dimension (1 + (N - 1)*abs(INCX))

INCX Input parameter.

INCX is INTEGER

storage spacing between elements of CX

CY Input and output parameter.

CY is COMPLEX

CY is an array, dimension (1 + (N - 1)*abs(INCY))

INCY Input parameter.

INCY is INTEGER

storage spacing between elements of CY

Related Information

For this routine in other precisions, please see *dswap*, *sswap* and *zswap*. It also exists with a native C interface as *cblas_cswap*.

3.2.11 dasum

```
DASUM takes the sum of the absolute values.
```

Syntax

Fortran specification:

```
use armpl_library

double precision function dasum(N, DX, INCX)
```

C specification:

```
#include "armpl.h"

double dasum_(const armpl_int_t *n, const double *x,
              const armpl_int_t *incx);
```

Parameters

N Input parameter.

N is INTEGER

number of elements in input vector(s)

DX Input parameter.

DX is DOUBLE PRECISION

DX is an array, dimension (1 + (N - 1)*abs(INCX))

INCX Input parameter.

INCX is INTEGER

storage spacing between elements of DX

Related Information

For this routine in other precisions, please see *sasum*. It also exists with a native C interface as *cblas_dasum*.

3.2.12 daxpy

daxpy performs the following vector update:

$y \leftarrow \alpha x + y$

Syntax

Fortran specification:

```
use armpl_library

subroutine daxpy(N, DA, DX, INCX, DY, INCY)
```

C specification:

```
#include "armpl.h"

void daxpy_(const armpl_int_t *n, const double *alpha, const double *x,
            const armpl_int_t *incx, double *y, const armpl_int_t *incy);
```

Parameters

N Input parameter.

N is INTEGER

N is the number of elements in the input vectors DX and DY.

DA Input parameter.

DA is DOUBLE PRECISION

DA specifies the scalar α.

DX Input parameter.

DX is DOUBLE PRECISION

DX is the array x, dimension (1 + (N - 1)*abs(INCX)).

INCX Input parameter.

INCX is INTEGER

INCX is the storage spacing between elements of DX.

DY Input and output parameter.

DY is DOUBLE PRECISION

DY is the array y, dimension (1 + (N - 1)*abs(INCY)).

INCY Input parameter.

INCY is INTEGER

INCY is the storage spacing between elements of DY.

Related Information

For this routine in other precisions, please see *caxpy*, *saxpy* and *zaxpy*. It also exists with a native C interface as *cblas_daxpy*.

3.2.13 dcopy

dcopy copies a vector, x, to a vector, y.

Syntax

Fortran specification:

```
use armpl_library

subroutine dcopy(N, DX, INCX, DY, INCY)
```

C specification:

```
#include "armpl.h"

void dcopy_(const armpl_int_t *n, const double *x, const armpl_int_t *incx,
            double *y, const armpl_int_t *incy);
```

Parameters

N Input parameter.

N is INTEGER

N is the number of elements in input vectors.

DX Input parameter.

DX is DOUBLE PRECISION

DX is the array x, dimension (1 + (N - 1)*abs(INCX)).

INCX Input parameter.

INCX is INTEGER

INCX is the storage spacing between elements of DX.

DY Output parameter.

DY is DOUBLE PRECISION

DY is the array y, dimension (1 + (N - 1)*abs(INCY)).

INCY Input parameter.

INCY is INTEGER

INCY is the storage spacing between elements of DY.

Related Information

For this routine in other precisions, please see *ccopy*, *scopy* and *zcopy*. It also exists with a native C interface as *cblas_dcopy*.

3.2.14 ddot

ddot forms the dot product of two vectors.

$$ddot = X^T \cdot Y$$

Syntax

Fortran specification:

```
use armpl_library

double precision function ddot(N, DX, INCX, DY, INCY)
```

C specification:

```
#include "armpl.h"

double ddot_(const armpl_int_t *n, const double *x, const armpl_int_t *incx,
             const double *y, const armpl_int_t *incy);
```

Parameters

N Input parameter.

N is INTEGER

number of elements in input vector(s)

DX Input parameter.

DX is DOUBLE PRECISION

DX is an array, dimension (1 + (N - 1)*abs(INCX))

INCX Input parameter.

INCX is INTEGER

storage spacing between elements of DX

DY Input parameter.

DY is DOUBLE PRECISION

DY is an array, dimension (1 + (N - 1)*abs(INCY))

INCY Input parameter.

INCY is INTEGER

storage spacing between elements of DY

Related Information

For this routine in other precisions, please see *sdot*. It also exists with a native C interface as *cblas_ddot*.

3.2.15 dnrm2

dnrm2 returns the euclidean norm of a vector via the function name, so that

```
DNRM2 := sqrt( x'*x )
```

Syntax

Fortran specification:

```
use armpl_library

double precision function dnrm2(N, X, INCX)
```

C specification:

```
#include "armpl.h"

double dnrm2_(const armpl_int_t *n, const double *x,
              const armpl_int_t *incx);
```

Parameters

N Input parameter.

N is INTEGER

number of elements in input vector(s)

X Input parameter.

X is DOUBLE PRECISION

X is an array, dimension (1 + (N - 1)*abs(INCX))

INCX Input parameter.

INCX is INTEGER

storage spacing between elements of DX

Related Information

For this routine in other precisions, please see *snrm2*. It also exists with a native C interface as *cblas_dnrm2*.

3.2.16 drot

```
DROT applies a plane rotation.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine drot(N, DX, INCX, DY, INCY, C, S)
```

C specification:

```
#include "armpl.h"

void drot_(const armpl_int_t *n, double *x, const armpl_int_t *incx,
           double *y, const armpl_int_t *incy, const double *c,
           const double *s);
```

Parameters

N Input parameter.

N is INTEGER

number of elements in input vector(s)

DX Input and output parameter.

DX is DOUBLE PRECISION

DX is an array, dimension (1 + (N - 1)*abs(INCX))

INCX Input parameter.

INCX is INTEGER

storage spacing between elements of DX

DY Input and output parameter.

DY is DOUBLE PRECISION

DY is an array, dimension (1 + (N - 1)*abs(INCY))

INCY Input parameter.

INCY is INTEGER

storage spacing between elements of DY

C Input parameter.

C is DOUBLE PRECISION

S Input parameter.

S is DOUBLE PRECISION

Related Information

For this routine in other precisions, please see *crot*, *srot* and *zrot*. It also exists with a native C interface as *cblas_drot*.

3.2.17 drotg

DROTG construct givens plane rotation.

Syntax

Fortran specification:

```
use armpl_library

subroutine drotg(DA, DB, C, S)
```

C specification:

```
#include "armpl.h"

void drotg_(double *a, double *b, double *c, double *s);
```

Parameters

DA Input parameter.

DA is DOUBLE PRECISION

DB Input parameter.

DB is DOUBLE PRECISION

C Output parameter.

C is DOUBLE PRECISION

S Output parameter.

S is DOUBLE PRECISION

Related Information

For this routine in other precisions, please see *crotg*, *srotg* and *zrotg*. It also exists with a native C interface as *cblas_drotg*.

3.2.18 drotm

```
APPLY THE MODIFIED GIVENS TRANSFORMATION, H, TO THE 2 BY N MATRIX

   (DX**T) , WHERE **T INDICATES TRANSPOSE. THE ELEMENTS OF DX ARE IN
   (DY**T)

   DX(LX+I*INCX), I = 0 TO N-1, WHERE LX = 1 IF INCX .GE. 0, ELSE
   LX = (-INCX)*N, AND SIMILARLY FOR SY USING LY AND INCY.
   WITH DPARAM(1)=DFLAG, H HAS ONE OF THE FOLLOWING FORMS..

   DFLAG=-1.D0     DFLAG=0.D0        DFLAG=1.D0     DFLAG=-2.D0

     (DH11  DH12)    (1.D0  DH12)    (DH11  1.D0)    (1.D0  0.D0)
   H=(          )    (          )    (          )    (          )
     (DH21  DH22),   (DH21  1.D0),   (-1.D0 DH22),   (0.D0  1.D0).
   SEE DROTMG FOR A DESCRIPTION OF DATA STORAGE IN DPARAM.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine drotm(N, DX, INCX, DY, INCY, DPARAM)
```

C specification:

```
#include "armpl.h"

void drotm_(const armpl_int_t *n, double *x, const armpl_int_t *incx,
            double *y, const armpl_int_t *incy, const double *param);
```

Parameters

N Input parameter.

N is INTEGER

number of elements in input vector(s)

DX Input and output parameter.

DX is DOUBLE PRECISION

DX is an array, dimension (1 + (N - 1)*abs(INCX))

INCX Input parameter.

INCX is INTEGER

storage spacing between elements of DX

DY Input and output parameter.

DY is DOUBLE PRECISION

DY is an array, dimension (1 + (N - 1)*abs(INCY))

INCY Input parameter.

INCY is INTEGER

storage spacing between elements of DY

DPARAM Input parameter.

DPARAM is DOUBLE PRECISION

DPARAM is an array, dimension (5). DPARAM(1)=DFLAG DPARAM(2)=DH11 DPARAM(3)=DH21 DPARAM(4)=DH12 DPARAM(5)=DH22

Related Information

For this routine in other precisions, please see *srotm*. It also exists with a native C interface as *cblas_drotm*.

3.2.19 drotmg

```
CONSTRUCT THE MODIFIED GIVENS TRANSFORMATION MATRIX H WHICH ZEROS
   THE SECOND COMPONENT OF THE 2-VECTOR  (DSQRT(DD1)*DX1,DSQRT(DD2)*>     DY2)**T.
   WITH DPARAM(1)=DFLAG, H HAS ONE OF THE FOLLOWING FORMS..

   DFLAG=-1.D0     DFLAG=0.D0        DFLAG=1.D0     DFLAG=-2.D0

     (DH11  DH12)    (1.D0  DH12)    (DH11  1.D0)    (1.D0  0.D0)
   H=(          )    (          )    (          )    (          )
     (DH21  DH22),   (DH21  1.D0),   (-1.D0 DH22),   (0.D0  1.D0).
   LOCATIONS 2-4 OF DPARAM CONTAIN DH11, DH21, DH12, AND DH22
   RESPECTIVELY. (VALUES OF 1.D0, -1.D0, OR 0.D0 IMPLIED BY THE
   VALUE OF DPARAM(1) ARE NOT STORED IN DPARAM.)

   THE VALUES OF GAMSQ AND RGAMSQ SET IN THE DATA STATEMENT MAY BE
   INEXACT.  THIS IS OK AS THEY ARE ONLY USED FOR TESTING THE SIZE
   OF DD1 AND DD2.  ALL ACTUAL SCALING OF DATA IS DONE USING GAM.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine drotmg(DD1, DD2, DX1, DY1, DPARAM)
```

C specification:

```
#include "armpl.h"

void drotmg_(double *d1, double *d2, double *b1, const double *b2,
             double *param);
```

Parameters

DD1 Input and output parameter.

DD1 is DOUBLE PRECISION

DD2 Input and output parameter.

DD2 is DOUBLE PRECISION

DX1 Input and output parameter.

DX1 is DOUBLE PRECISION

DY1 Input parameter.

DY1 is DOUBLE PRECISION

DPARAM Output parameter.

DPARAM is DOUBLE PRECISION

DPARAM is an array, dimension (5). DPARAM(1)=DFLAG DPARAM(2)=DH11 DPARAM(3)=DH21 DPARAM(4)=DH12 DPARAM(5)=DH22

Related Information

For this routine in other precisions, please see *srotmg*. It also exists with a native C interface as *cblas_drotmg*.

3.2.20 dscal

```
DSCAL scales a vector by a constant.
   uses unrolled loops for increment equal to 1.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dscal(N, DA, DX, INCX)
```

C specification:

```
#include "armpl.h"

void dscal_(const armpl_int_t *n, const double *alpha, double *x,
            const armpl_int_t *incx);
```

Parameters

N Input parameter.

N is INTEGER

number of elements in input vector(s)

DA Input parameter.

DA is DOUBLE PRECISION

On entry, DA specifies the scalar alpha.

DX Input and output parameter.

DX is DOUBLE PRECISION

DX is an array, dimension (1 + (N - 1)*abs(INCX))

INCX Input parameter.

INCX is INTEGER

storage spacing between elements of DX

Related Information

For this routine in other precisions, please see *cscal*, *sscal* and *zscal*. It also exists with a native C interface as *cblas_dscal*.

3.2.21 dsdot

Compute the inner product of two vectors with extended precision accumulation and result.

Returns D.P. dot product accumulated in D.P., for S.P. SX and SY `dsdot` = sum for I = 0 to N-1 of SX(LX+I*INCX) * SY(LY+I*INCY), where LX = 1 if INCX .GE. 0, else LX = 1+(1-N)*INCX, and LY is defined in a similar way using INCY.

Syntax

Fortran specification:

```
use armpl_library

double precision function dsdot(N, SX, INCX, SY, INCY)
```

C specification:

```
#include "armpl.h"

double dsdot_(const armpl_int_t *n, const float *sx, const armpl_int_t *incx,
              const float *sy, const armpl_int_t *incy);
```

Parameters

N Input parameter.

N is INTEGER

number of elements in input vector(s)

SX Input parameter.

SX is REAL

SX is an array, dimension(N). single precision vector with N elements

INCX Input parameter.

INCX is INTEGER

storage spacing between elements of SX

SY Input parameter.

SY is REAL

SY is an array, dimension(N). single precision vector with N elements

INCY Input parameter.

INCY is INTEGER

storage spacing between elements of SY

Related Information

It also exists with a native C interface as *cblas_dsdot*.

3.2.22 dswap

```
DSWAP interchanges two vectors.
   uses unrolled loops for increments equal to 1.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dswap(N, DX, INCX, DY, INCY)
```

C specification:

```
#include "armpl.h"

void dswap_(const armpl_int_t *n, double *x, const armpl_int_t *incx,
            double *y, const armpl_int_t *incy);
```

Parameters

N Input parameter.

N is INTEGER

number of elements in input vector(s)

DX Input and output parameter.

DX is DOUBLE PRECISION

DX is an array, dimension (1 + (N - 1)*abs(INCX))

INCX Input parameter.

INCX is INTEGER

storage spacing between elements of DX

DY Input and output parameter.

DY is DOUBLE PRECISION

DY is an array, dimension (1 + (N - 1)*abs(INCY))

INCY Input parameter.

INCY is INTEGER

storage spacing between elements of DY

Related Information

For this routine in other precisions, please see *cswap*, *sswap* and *zswap*. It also exists with a native C interface as *cblas_dswap*.

3.2.23 dzasum

DZASUM takes the sum of the (|Re(.)| + |Im(.)|)'s of a complex vector and returns a single precision result.

Syntax

Fortran specification:

```
use armpl_library

double precision function dzasum(N, ZX, INCX)
```

C specification:

```
#include "armpl.h"

double dzasum_(const armpl_int_t *n, const armpl_doublecomplex_t *x,
               const armpl_int_t *incx);
```

Parameters

N Input parameter.

N is INTEGER

number of elements in input vector(s)

ZX Input and output parameter.

ZX is COMPLEX*16

ZX is an array, dimension (1 + (N - 1)*abs(INCX))

INCX Input parameter.

INCX is INTEGER

storage spacing between elements of ZX

Related Information

It also exists with a native C interface as *cblas_dzasum*.

3.2.24 dznrm2

dznrm2 returns the euclidean norm of a vector via the function name, so that

```
DZNRM2 := sqrt( x**H*x )
```

Syntax

Fortran specification:

```
use armpl_library

double precision function dznrm2(N, X, INCX)
```

C specification:

```
#include "armpl.h"

double dznrm2_(const armpl_int_t *n, const armpl_doublecomplex_t *x,
               const armpl_int_t *incx);
```

Parameters

N Input parameter.

N is INTEGER

number of elements in input vector(s)

X Input parameter.

X is COMPLEX*16

X is an array, dimension (N). complex vector with N elements

INCX Input parameter.

INCX is INTEGER

storage spacing between elements of X

Related Information

It also exists with a native C interface as *cblas_dznrm2*.

3.2.25 icamax

```
ICAMAX finds the index of the first element having maximum |Re(.)| + |Im(.)|
```

Syntax

Fortran specification:

```
use armpl_library

integer function icamax(N, CX, INCX)
```

C specification:

```
#include "armpl.h"

armpl_int_t icamax_(const armpl_int_t *n, const armpl_singlecomplex_t *x,
                    onst armpl_int_t *incx);
```

Parameters

N Input parameter.

N is INTEGER

number of elements in input vector(s)

CX Input parameter.

CX is COMPLEX

CX is an array, dimension (1 + (N - 1)*abs(INCX))

INCX Input parameter.

INCX is INTEGER

storage spacing between elements of SX

Related Information

It also exists with a native C interface as *cblas_icamax*.

3.2.26 idamax

```
IDAMAX finds the index of the first element having maximum absolute value.
```

Syntax

Fortran specification:

```
use armpl_library

integer function idamax(N, DX, INCX)
```

C specification:

```
#include "armpl.h"

armpl_int_t idamax_(const armpl_int_t *n, const double *x,
                    onst armpl_int_t *incx);
```

Parameters

N Input parameter.

N is INTEGER

number of elements in input vector(s)

DX Input parameter.

DX is DOUBLE PRECISION

DX is an array, dimension (1 + (N - 1)*abs(INCX))

INCX Input parameter.

INCX is INTEGER

storage spacing between elements of SX

Related Information

It also exists with a native C interface as *cblas_idamax*.

3.2.27 isamax

```
ISAMAX finds the index of the first element having maximum absolute value.
```

Syntax

Fortran specification:

```
use armpl_library

integer function isamax(N, SX, INCX)
```

C specification:

```
#include "armpl.h"

armpl_int_t isamax_(const armpl_int_t *n, const float *x,
                    onst armpl_int_t *incx);
```

Parameters

N Input parameter.

N is INTEGER

number of elements in input vector(s)

SX Input parameter.

SX is REAL

SX is an array, dimension (1 + (N - 1)*abs(INCX))

INCX Input parameter.

INCX is INTEGER

storage spacing between elements of SX

Related Information

It also exists with a native C interface as *cblas_isamax*.

3.2.28 izamax

```
IZAMAX finds the index of the first element having maximum |Re(.)| + |Im(.)|
```

Syntax

Fortran specification:

```
use armpl_library

integer function izamax(N, ZX, INCX)
```

C specification:

```
#include "armpl.h"

armpl_int_t izamax_(const armpl_int_t *n, const armpl_doublecomplex_t *x,
                    onst armpl_int_t *incx);
```

Parameters

N Input parameter.

N is INTEGER

number of elements in input vector(s)

ZX Input parameter.

ZX is COMPLEX*16

ZX is an array, dimension (1 + (N - 1)*abs(INCX))

INCX Input parameter.

INCX is INTEGER

storage spacing between elements of SX

Related Information

It also exists with a native C interface as *cblas_izamax*.

3.2.29 sasum

```
SASUM takes the sum of the absolute values.
   uses unrolled loops for increment equal to one.
```

Syntax

Fortran specification:

```
use armpl_library

real function sasum(N, SX, INCX)
```

C specification:

```
#include "armpl.h"

float sasum_(const armpl_int_t *n, const float *x, const armpl_int_t *incx);
```

Parameters

N Input parameter.

N is INTEGER

number of elements in input vector(s)

SX Input parameter.

SX is REAL

SX is an array, dimension (1 + (N - 1)*abs(INCX))

INCX Input parameter.

INCX is INTEGER

storage spacing between elements of SX

Related Information

For this routine in other precisions, please see *dasum*. It also exists with a native C interface as *cblas_sasum*.

3.2.30 saxpy

caxpy performs the following vector update:

$$y \leftarrow \alpha x + y$$

Syntax

Fortran specification:

```
use armpl_library

subroutine saxpy(N, SA, SX, INCX, SY, INCY)
```

C specification:

```
#include "armpl.h"

void saxpy_(const armpl_int_t *n, const float *alpha, const float *x,
            const armpl_int_t *incx, float *y, const armpl_int_t *incy);
```

Parameters

N Input parameter.

N is INTEGER

N is the number of elements in the input vectors SX and SY.

SA Input parameter.

SA is REAL

SA specifies the scalar $alpha$.

SX Input parameter.

SX is REAL

SX is the array x, dimension (1 + (N - 1)*abs(INCX)).

INCX Input parameter.

INCX is INTEGER

INCX is the storage spacing between elements of SX.

SY Input and output parameter.

SY is REAL

SY is the array y, dimension (1 + (N - 1)*abs(INCY)).

INCY Input parameter.

INCY is INTEGER

INCY is the storage spacing between elements of SY.

Related Information

For this routine in other precisions, please see *caxpy*, *daxpy* and *zaxpy*. It also exists with a native C interface as *cblas_saxpy*.

3.2.31 scasum

SCASUM takes the sum of the (|Re(.)| + |Im(.)|)'s of a complex vector and returns a single precision result.

Syntax

Fortran specification:

```
use armpl_library

real function scasum(N, CX, INCX)
```

C specification:

```
#include "armpl.h"

float scasum_(const armpl_int_t *n, const armpl_singlecomplex_t *x,
              const armpl_int_t *incx);
```

Parameters

N Input parameter.

N is INTEGER

number of elements in input vector(s)

CX Input and output parameter.

CX is COMPLEX

CX is an array, dimension (1 + (N - 1)*abs(INCX))

INCX Input parameter.

INCX is INTEGER

storage spacing between elements of SX

Related Information

It also exists with a native C interface as *cblas_scasum*.

3.2.32 scnrm2

scnrm2 returns the euclidean norm of a vector via the function name, so that

```
SCNRM2 := sqrt( x**H*x )
```

Syntax

Fortran specification:

```
use armpl_library

real function scnrm2(N, X, INCX)
```

C specification:

```
#include "armpl.h"

float scnrm2_(const armpl_int_t *n, const armpl_singlecomplex_t *x,
              const armpl_int_t *incx);
```

Parameters

N Input parameter.

N is INTEGER

number of elements in input vector(s)

X Input parameter.

X is COMPLEX

X is an array, dimension (N). complex vector with N elements

INCX Input parameter.

INCX is INTEGER

storage spacing between elements of X

Related Information

It also exists with a native C interface as *cblas_scnrm2*.

3.2.33 scopy

scopy copies a vector, x, to a vector, y.

Syntax

Fortran specification:

```
use armpl_library

subroutine scopy(N, SX, INCX, SY, INCY)
```

C specification:

```
#include "armpl.h"

void scopy_(const armpl_int_t *n, const float *x, const armpl_int_t *incx,
            float *y, const armpl_int_t *incy);
```

Parameters

N Input parameter.

N is INTEGER

N is the number of elements in input vectors.

SX Input parameter.

SX is REAL

SX is the array x, dimension (1 + (N - 1)*abs(INCX)).

INCX Input parameter.

INCX is INTEGER

INCX is the storage spacing between elements of SX.

SY Output parameter.

SY is REAL

SY is the array y, dimension (1 + (N - 1)*abs(INCY)).

INCY Input parameter.

INCY is INTEGER

INCY is the storage spacing between elements of SY.

Related Information

For this routine in other precisions, please see *ccopy*, *dcopy* and *zcopy*. It also exists with a native C interface as *cblas_scopy*.

3.2.34 sdot

sdot forms the dot product of two vectors.

$$SDOT = X^T \cdot Y$$

Syntax

Fortran specification:

```
use armpl_library

real function sdot(N, SX, INCX, SY, INCY)
```

C specification:

```
#include "armpl.h"

float sdot_(const armpl_int_t *n, const float *x, const armpl_int_t *incx,
            const float *y, const armpl_int_t *incy);
```

Parameters

N Input parameter.

N is INTEGER

number of elements in input vector(s)

SX Input parameter.

SX is REAL

SX is an array, dimension (1 + (N - 1)*abs(INCX))

INCX Input parameter.

INCX is INTEGER

storage spacing between elements of SX

SY Input parameter.

SY is REAL

SY is an array, dimension (1 + (N - 1)*abs(INCY))

INCY Input parameter.

INCY is INTEGER

storage spacing between elements of SY

Related Information

For this routine in other precisions, please see *ddot*. It also exists with a native C interface as *cblas_sdot*.

3.2.35 sdsdot

Syntax

Fortran specification:

```
use armpl_library

real function sdsdot(N, SB, SX, INCX, SY, INCY)
```

C specification:

```
#include "armpl.h"

float sdsdot_(const armpl_int_t *n, const float *b, const float *x,
              const armpl_int_t *incx, const float *y,
              const armpl_int_t *incy);
```

Parameters

N Input parameter.

N is INTEGER

number of elements in input vector(s)

SB Input parameter.

SB is REAL

single precision scalar to be added to inner product

SX Input parameter.

SX is REAL

SX is an array, dimension (1 + (N - 1)*abs(INCX)). single precision vector with N elements

INCX Input parameter.

INCX is INTEGER

storage spacing between elements of SX

SY Input parameter.

SY is REAL

SY is an array, dimension (1 + (N - 1)*abs(INCX)). single precision vector with N elements

INCY Input parameter.

INCY is INTEGER

storage spacing between elements of SY

Related Information

It also exists with a native C interface as *cblas_sdsdot*.

3.2.36 snrm2

snrm2 returns the euclidean norm of a vector via the function name, so that

```
SNRM2 := sqrt( x'*x ).
```

Syntax

Fortran specification:

```
use armpl_library

real function snrm2(N, X, INCX)
```

C specification:

```
#include "armpl.h"

float snrm2_(const armpl_int_t *n, const float *x, const armpl_int_t *incx);
```

Parameters

N Input parameter.

N is INTEGER

number of elements in input vector(s)

X Input parameter.

X is REAL

X is an array, dimension (1 + (N - 1)*abs(INCX))

INCX Input parameter.

INCX is INTEGER

storage spacing between elements of SX

Related Information

For this routine in other precisions, please see *dnrm2*. It also exists with a native C interface as *cblas_snrm2*.

3.2.37 srot

applies a plane rotation.

Syntax

Fortran specification:

```
use armpl_library

subroutine srot(N, SX, INCX, SY, INCY, C, S)
```

C specification:

```
#include "armpl.h"

void srot_(const armpl_int_t *n, float *x, const armpl_int_t *incx, float *y,
           const armpl_int_t *incy, const float *c, const float *s);
```

Parameters

N Input parameter.

N is INTEGER

number of elements in input vector(s)

SX Input and output parameter.

SX is REAL

SX is an array, dimension (1 + (N - 1)*abs(INCX))

INCX Input parameter.

INCX is INTEGER

storage spacing between elements of SX

SY Input and output parameter.

SY is REAL

SY is an array, dimension (1 + (N - 1)*abs(INCY))

INCY Input parameter.

INCY is INTEGER

storage spacing between elements of SY

C Input parameter.

C is REAL

S Input parameter.

S is REAL

Related Information

For this routine in other precisions, please see *crot*, *drot* and *zrot*. It also exists with a native C interface as *cblas_srot*.

3.2.38 srotg

SROTG construct givens plane rotation.

Syntax

Fortran specification:

```
use armpl_library

subroutine srotg(SA, SB, C, S)
```

C specification:

```
#include "armpl.h"

void srotg_(float *a, float *b, float *c, float *s);
```

Parameters

SA Input parameter.

SA is REAL

SB Input parameter.

SB is REAL

C Output parameter.

C is REAL

S Output parameter.

S is REAL

Related Information

For this routine in other precisions, please see *crotg*, *drotg* and *zrotg*. It also exists with a native C interface as *cblas_srotg*.

3.2.39 srotm

```
APPLY THE MODIFIED GIVENS TRANSFORMATION, H, TO THE 2 BY N MATRIX

   (SX**T) , WHERE **T INDICATES TRANSPOSE. THE ELEMENTS OF SX ARE IN
   (SX**T)

   SX(LX+I*INCX), I = 0 TO N-1, WHERE LX = 1 IF INCX .GE. 0, ELSE
   LX = (-INCX)*N, AND SIMILARLY FOR SY USING USING LY AND INCY.
   WITH SPARAM(1)=SFLAG, H HAS ONE OF THE FOLLOWING FORMS..

   SFLAG=-1.E0     SFLAG=0.E0         SFLAG=1.E0     SFLAG=-2.E0

     (SH11  SH12)    (1.E0  SH12)    (SH11  1.E0)    (1.E0  0.E0)
   H=(          )    (          )    (          )    (          )
     (SH21  SH22),   (SH21  1.E0),   (-1.E0 SH22),   (0.E0  1.E0).
   SEE  SROTMG FOR A DESCRIPTION OF DATA STORAGE IN SPARAM.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine srotm(N, SX, INCX, SY, INCY, SPARAM)
```

C specification:

```
#include "armpl.h"

void srotm_(const armpl_int_t *n, float *x, const armpl_int_t *incx, float *y,
            const armpl_int_t *incy, const float *param);
```

Parameters

N Input parameter.

N is INTEGER

number of elements in input vector(s)

SX Input and output parameter.

SX is REAL

SX is an array, dimension (1 + (N - 1)*abs(INCX))

INCX Input parameter.

INCX is INTEGER

storage spacing between elements of SX

SY Input and output parameter.

SY is REAL

SY is an array, dimension (1 + (N - 1)*abs(INCY))

INCY Input parameter.

INCY is INTEGER

storage spacing between elements of SY

SPARAM Input parameter.

SPARAM is REAL

SPARAM is an array, dimension (5). SPARAM(1)=SFLAG SPARAM(2)=SH11 SPARAM(3)=SH21 SPARAM(4)=SH12 SPARAM(5)=SH22

Related Information

For this routine in other precisions, please see *drotm*. It also exists with a native C interface as *cblas_srotm*.

3.2.40 srotmg

```
CONSTRUCT THE MODIFIED GIVENS TRANSFORMATION MATRIX H WHICH ZEROS
   THE SECOND COMPONENT OF THE 2-VECTOR  (SQRT(SD1)*SX1,SQRT(SD2)*>     SY2)**T.
   WITH SPARAM(1)=SFLAG, H HAS ONE OF THE FOLLOWING FORMS..

   SFLAG=-1.E0     SFLAG=0.E0         SFLAG=1.E0     SFLAG=-2.E0

     (SH11  SH12)     (1.E0  SH12)     (SH11  1.E0)     (1.E0  0.E0)
   H=(          )     (          )     (          )     (          )
     (SH21  SH22),    (SH21  1.E0),    (-1.E0 SH22),    (0.E0  1.E0).
   LOCATIONS 2-4 OF SPARAM CONTAIN SH11,SH21,SH12, AND SH22
   RESPECTIVELY. (VALUES OF 1.E0, -1.E0, OR 0.E0 IMPLIED BY THE
   VALUE OF SPARAM(1) ARE NOT STORED IN SPARAM.)

   THE VALUES OF GAMSQ AND RGAMSQ SET IN THE DATA STATEMENT MAY BE
   INEXACT.  THIS IS OK AS THEY ARE ONLY USED FOR TESTING THE SIZE
   OF SD1 AND SD2.  ALL ACTUAL SCALING OF DATA IS DONE USING GAM.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine srotmg(SD1, SD2, SX1, SY1, SPARAM)
```

C specification:

```
#include "armpl.h"

void srotmg_(float *d1, float *d2, float *b1, const float *b2, float *param);
```

Parameters

SD1 Input and output parameter.

SD1 is REAL

SD2 Input and output parameter.

SD2 is REAL

SX1 Input and output parameter.

SX1 is REAL

SY1 Input parameter.

SY1 is REAL

SPARAM Output parameter.

SPARAM is REAL

SPARAM is an array, dimension (5). SPARAM(1)=SFLAG SPARAM(2)=SH11 SPARAM(3)=SH21 SPARAM(4)=SH12 SPARAM(5)=SH22

Related Information

For this routine in other precisions, please see *drotmg*. It also exists with a native C interface as *cblas_srotmg*.

3.2.41 sscal

```
SSCAL scales a vector by a constant.
   uses unrolled loops for increment equal to 1.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine sscal(N, SA, SX, INCX)
```

C specification:

```
#include "armpl.h"

void sscal_(const armpl_int_t *n, const float *alpha, float *x,
            const armpl_int_t *incx);
```

Parameters

N Input parameter.

N is INTEGER

number of elements in input vector(s)

SA Input parameter.

SA is REAL

On entry, SA specifies the scalar alpha.

SX Input and output parameter.

SX is REAL

SX is an array, dimension (1 + (N - 1)*abs(INCX))

INCX Input parameter.

INCX is INTEGER

storage spacing between elements of SX

Related Information

For this routine in other precisions, please see *cscal*, *dscal* and *zscal*. It also exists with a native C interface as *cblas_sscal*.

3.2.42 sswap

```
SSWAP interchanges two vectors.
   uses unrolled loops for increments equal to 1.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine sswap(N, SX, INCX, SY, INCY)
```

C specification:

```
#include "armpl.h"

void sswap_(const armpl_int_t *n, float *x, const armpl_int_t *incx, float *y,
            const armpl_int_t *incy);
```

Parameters

N Input parameter.

N is INTEGER

number of elements in input vector(s)

SX Input and output parameter.

SX is REAL

SX is an array, dimension (1 + (N - 1)*abs(INCX))

INCX Input parameter.

INCX is INTEGER

storage spacing between elements of SX

SY Input and output parameter.

SY is REAL

SY is an array, dimension (1 + (N - 1)*abs(INCY))

INCY Input parameter.

INCY is INTEGER

storage spacing between elements of SY

Related Information

For this routine in other precisions, please see *cswap*, *dswap* and *zswap*. It also exists with a native C interface as *cblas_sswap*.

3.2.43 zaxpy

caxpy performs the following vector update:

$y \leftarrow \alpha x + y$

Syntax

Fortran specification:

```
use armpl_library

subroutine zaxpy(N, ZA, ZX, INCX, ZY, INCY)
```

C specification:

```
#include "armpl.h"

void zaxpy_(const armpl_int_t *n, const armpl_doublecomplex_t *alpha,
            const armpl_doublecomplex_t *x, const armpl_int_t *incx,
            armpl_doublecomplex_t *y, const armpl_int_t *incy);
```

Parameters

N Input parameter.

N is INTEGER

N is the number of elements in the input vectors ZX and ZY.

ZA Input parameter.

ZA is COMPLEX*16

ZA specifies the scalar α.

ZX Input parameter.

ZX is COMPLEX*16

ZX is the array x, dimension (1 + (N - 1)*abs(INCX)).

INCX Input parameter.

INCX is INTEGER

INCX is the storage spacing between elements of ZX.

ZY Input and output parameter.

ZY is COMPLEX*16

ZY is the array y, dimension (1 + (N - 1)*abs(INCY)).

INCY Input parameter.

INCY is INTEGER

INCY is the storage spacing between elements of ZY.

Related Information

For this routine in other precisions, please see *caxpy*, *daxpy* and *saxpy*. It also exists with a native C interface as *cblas_zaxpy*.

3.2.44 zcopy

zcopy copies a vector, x, to a vector, y.

Syntax

Fortran specification:

```
use armpl_library

subroutine zcopy(N, ZX, INCX, ZY, INCY)
```

C specification:

```
#include "armpl.h"

void zcopy_(const armpl_int_t *n, const armpl_doublecomplex_t *x,
            const armpl_int_t *incx, armpl_doublecomplex_t *y,
            const armpl_int_t *incy);
```

Parameters

N Input parameter.

N is INTEGER

N is the number of elements in input vectors.

ZX Input parameter.

ZX is COMPLEX*16

ZX is the array x, dimension (1 + (N - 1)*abs(INCX)).

INCX Input parameter.

INCX is INTEGER

INCX is the storage spacing between elements of ZX.

ZY Output parameter.

ZY is COMPLEX*16

ZY is the array y, dimension (1 + (N - 1)*abs(INCY)).

INCY Input parameter.

INCY is INTEGER

INCY is the storage spacing between elements of ZY.

Related Information

For this routine in other precisions, please see *ccopy*, *dcopy* and *scopy*. It also exists with a native C interface as *cblas_zcopy*.

3.2.45 zdotc

zdotc forms the dot product of two complex vectors.

$$ZDOTC = X^H \cdot Y$$

Syntax

Fortran specification:

```
use armpl_library

complex*16 function zdotc(N, ZX, INCX, ZY, INCY)
```

C specification:

```
#include "armpl.h"

DOUBLECOMPLEX_RET_VALUE zdotc_(const armpl_int_t *n,
                               const armpl_doublecomplex_t *x,
                               const armpl_int_t *incx,
                               const armpl_doublecomplex_t *y,
                               const armpl_int_t *incy);
```

Returns

This returns the result of the dot product, ZDOTC.

Parameters

N Input parameter.

N is INTEGER

number of elements in input vector(s)

ZX Input parameter.

ZX is REAL

ZX is an array, dimension (1 + (N - 1)*abs(INCX))

INCX Input parameter.

INCX is INTEGER

storage spacing between elements of ZX

ZY Input parameter.

ZY is REAL

ZY is an array, dimension (1 + (N - 1)*abs(INCY))

INCY Input parameter.

INCY is INTEGER

storage spacing between elements of ZY

Related Information

For this routine in other precisions, please see *cdotc*.

3.2.46 zdotu

zdotu forms the dot product of two complex vectors.

$$ZDOTU = X^T \cdot Y$$

Syntax

Fortran specification:

```
use armpl_library

complex*16 function zdotu(N, ZX, INCX, ZY, INCY)
```

C specification:

```
#include "armpl.h"

DOUBLECOMPLEX_RET_VALUE zdotu_(const armpl_int_t *n,
                               const armpl_doublecomplex_t *x,
                               const armpl_int_t *incx,
                               const armpl_doublecomplex_t *y,
                               const armpl_int_t *incy);
```

Returns

This returns the result of the dot product, ZDOTU.

Parameters

N Input parameter.

N is INTEGER

number of elements in input vector(s)

ZX Input parameter.

ZX is REAL

ZX is an array, dimension (1 + (N - 1)*abs(INCX))

INCX Input parameter.

INCX is INTEGER

storage spacing between elements of ZX

ZY Input parameter.

ZY is REAL

ZY is an array, dimension (1 + (N - 1)*abs(INCY))

INCY Input parameter.

INCY is INTEGER

storage spacing between elements of ZY

Related Information

For this routine in other precisions, please see *cdotu*.

3.2.47 zdrot

Applies a plane rotation, where the cos and sin (c and s) are real and the vectors cx and cy are complex. jack dongarra, linpack, 3/11/78.

Syntax

Fortran specification:

```
use armpl_library

subroutine zdrot(N, CX, INCX, CY, INCY, C, S)
```

C specification:

```
#include "armpl.h"

void zdrot_(const armpl_int_t *n, armpl_doublecomplex_t *x,
            const armpl_int_t *incx, armpl_doublecomplex_t *y,
            const armpl_int_t *incy, const double *c, const double *s);
```

Parameters

N Input parameter.

N is INTEGER

On entry, N specifies the order of the vectors cx and cy. N must be at least zero.

CX Input and output parameter.

CX is COMPLEX*16

CX is an array, dimension at least. (1 + (N - 1)*abs(INCX)). Before entry, the incremented array CX must contain the n element vector cx. On exit, CX is overwritten by the updated vector cx.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of CX. INCX must not be zero.

CY Input and output parameter.

CY is COMPLEX*16

CY is an array, dimension at least. (1 + (N - 1)*abs(INCY)). Before entry, the incremented array CY must contain the n element vector cy. On exit, CY is overwritten by the updated vector cy.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of CY. INCY must not be zero.

C Input parameter.

C is DOUBLE PRECISION

On entry, C specifies the cosine, cos.

S Input parameter.

S is DOUBLE PRECISION

On entry, S specifies the sine, sin.

Related Information

3.2.48 zdscal

ZDSCAL scales a vector by a constant.

Syntax

Fortran specification:

```
use armpl_library

subroutine zdscal(N, DA, ZX, INCX)
```

C specification:

```
#include "armpl.h"

void zdscal_(const armpl_int_t *n, const double *alpha,
             armpl_doublecomplex_t *x, const armpl_int_t *incx);
```

Parameters

N Input parameter.

N is INTEGER

number of elements in input vector(s)

DA Input parameter.

DA is DOUBLE PRECISION

On entry, DA specifies the scalar alpha.

ZX Input and output parameter.

ZX is COMPLEX*16

ZX is an array, dimension (1 + (N - 1)*abs(INCX))

INCX Input parameter.

INCX is INTEGER

storage spacing between elements of ZX

Related Information

It also exists with a native C interface as *cblas_zdscal*.

3.2.49 zrot

zrot applies a plane rotation, where the cos (C) is real and the sin (S) is complex, and the vectors CX and CY are complex.

Syntax

Fortran specification:

```
use armpl_library

subroutine zrot(N, CX, INCX, CY, INCY, C, S)
```

C specification:

```
#include "armpl.h"

void zrot_(const armpl_int_t *n, armpl_doublecomplex_t *cx,
           const armpl_int_t *incx, armpl_doublecomplex_t *cy,
           const armpl_int_t *incy, const double *c,
           const armpl_doublecomplex_t *s);
```

Parameters

N Input parameter.

N is INTEGER

The number of elements in the vectors CX and CY.

CX Input and output parameter.

CX is COMPLEX*16

CX is an array, dimension (N). On input, the vector X. On output, CX is overwritten with C*X + S*Y.

INCX Input parameter.

INCX is INTEGER

The increment between successive values of CY. INCX <> 0.

CY Input and output parameter.

CY is COMPLEX*16

CY is an array, dimension (N). On input, the vector Y. On output, CY is overwritten with -CONJG(S)*X + C*Y.

INCY Input parameter.

INCY is INTEGER

The increment between successive values of CY. INCX <> 0.

C Input parameter.

C is DOUBLE PRECISION

S Input parameter.

S is COMPLEX*16

C and S define a rotation [C S] [-conjg(S) C] where C* C + S*CONJG(S) = 1.0.

Related Information

For this routine in other precisions, please see *crot*, *drot* and *srot*.

3.2.50 zrotg

ZROTG determines a double complex Givens rotation.

Syntax

Fortran specification:

```
use armpl_library

subroutine zrotg(CA, CB, C, S)
```

C specification:

```
#include "armpl.h"

void zrotg_(armpl_doublecomplex_t *ca, const armpl_doublecomplex_t *cb,
            double *c, armpl_doublecomplex_t *s);
```

Parameters

CA Input parameter.

CA is COMPLEX*16

CB Input parameter.

CB is COMPLEX*16

C Output parameter.

C is DOUBLE PRECISION

S Output parameter.

S is COMPLEX*16

Related Information

For this routine in other precisions, please see *crotg*, *drotg* and *srotg*.

3.2.51 zscal

ZSCAL scales a vector by a constant.

Syntax

Fortran specification:

```
use armpl_library

subroutine zscal(N, ZA, ZX, INCX)
```

C specification:

```
#include "armpl.h"

void zscal_(const armpl_int_t *n, const armpl_doublecomplex_t *alpha,
            armpl_doublecomplex_t *x, const armpl_int_t *incx);
```

Parameters

N Input parameter.

N is INTEGER

number of elements in input vector(s)

ZA Input parameter.

ZA is COMPLEX*16

On entry, ZA specifies the scalar alpha.

ZX Input and output parameter.

ZX is COMPLEX*16

ZX is an array, dimension (1 + (N - 1)*abs(INCX))

INCX Input parameter.

INCX is INTEGER

storage spacing between elements of ZX

Related Information

For this routine in other precisions, please see *cscal*, *dscal* and *sscal*. It also exists with a native C interface as *cblas_zscal*.

3.2.52 zswap

```
ZSWAP interchanges two vectors.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zswap(N, ZX, INCX, ZY, INCY)
```

C specification:

```
#include "armpl.h"

void zswap_(const armpl_int_t *n, armpl_doublecomplex_t *x,
            const armpl_int_t *incx, armpl_doublecomplex_t *y,
            const armpl_int_t *incy);
```

Parameters

N Input parameter.

N is INTEGER

number of elements in input vector(s)

ZX Input and output parameter.

ZX is COMPLEX*16

ZX is an array, dimension (1 + (N - 1)*abs(INCX))

INCX Input parameter.

INCX is INTEGER

storage spacing between elements of ZX

ZY Input and output parameter.

ZY is COMPLEX*16

ZY is an array, dimension (1 + (N - 1)*abs(INCY))

INCY Input parameter.

INCY is INTEGER

storage spacing between elements of ZY

Related Information

For this routine in other precisions, please see *cswap*, *dswap* and *sswap*. It also exists with a native C interface as *cblas_zswap*.

Level 1 BLAS routines are memory bound. All functions are detailed in armpl.h.

3.3 BLAS level 2

3.3.1 cgbmv

cgbmv performs one of the matrix-vector operations

```
 y := alpha*A*x + beta*y,   or   y := alpha*A**T*x + beta*y,   or

y := alpha*A**H*x + beta*y,
```

where alpha and beta are scalars, x and y are vectors and A is an m by n band matrix, with kl sub-diagonals and ku super-diagonals.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgbmv(TRANS, M, N, KL, KU, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
```

C specification:

```
#include "armpl.h"

void cgbmv_(const char *trans, const armpl_int_t *m, const armpl_int_t *n,
            const armpl_int_t *kl, const armpl_int_t *ku,
            const armpl_singlecomplex_t *alpha,
            const armpl_singlecomplex_t *a, const armpl_int_t *lda,
            const armpl_singlecomplex_t *x, const armpl_int_t *incx,
            const armpl_singlecomplex_t *beta, armpl_singlecomplex_t *y,
            const armpl_int_t *incy, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' y := alpha*A*x + beta*y.

TRANS = 'T' or 't' y := alpha*A^T *x + beta*y.

TRANS = 'C' or 'c' y := alpha*A^H *x + beta*y.

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of the matrix A. M must be at least zero.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix A. N must be at least zero.

KL Input parameter.

KL is INTEGER

On entry, KL specifies the number of sub-diagonals of the matrix A. KL must satisfy 0 .le. KL.

KU Input parameter.

KU is INTEGER

On entry, KU specifies the number of super-diagonals of the matrix A. KU must satisfy 0 .le. KU.

ALPHA Input parameter.

ALPHA is COMPLEX

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). Before entry, the leading (kl + ku + 1) by n part of the array A must contain the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row (ku + 1) of the array, the first super-diagonal starting at position 2 in row ku, the first sub-diagonal starting at position 1 in row (ku + 2), and so on. Elements in the array A that do not correspond to elements in the band matrix (such as the top left ku by ku triangle) are not referenced. The following program segment will transfer a band matrix from conventional full matrix storage to band storage:

```
   DO 20, J = 1, N
      K = ``KU`` + 1 - J
      DO 10, I = MAX( 1, J - ``KU`` ), MIN( ``M``, J + ``KL`` )
         A( K + I, J ) = matrix( I, J )
10    CONTINUE
20 CONTINUE
```

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least (kl + ku + 1).

X Input parameter.

X is COMPLEX

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)) when TRANS = 'N' or 'n' and at least (1 + (m - 1)*abs(INCX)) otherwise. Before entry, the incremented array X must contain the vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

BETA Input parameter.

BETA is COMPLEX

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input.

Y Input and output parameter.

Y is COMPLEX

Y is an array, dimension at least. (1 + (m - 1)*abs(INCY)) when TRANS = 'N' or 'n' and at least (1 + (n - 1)*abs(INCY)) otherwise. Before entry, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

Related Information

For this routine in other precisions, please see *dgbmv*, *sgbmv* and *zgbmv*. It also exists with a native C interface as *cblas_cgbmv*.

3.3.2 cgemv

cgemv performs one of the matrix-vector operations

```
 y := alpha*A*x + beta*y,   or   y := alpha*A**T*x + beta*y,   or

y := alpha*A**H*x + beta*y,
```

where alpha and beta are scalars, x and y are vectors and A is an m by n matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgemv(TRANS, M, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
```

C specification:

```
#include "armpl.h"

void cgemv_(const char *trans, const armpl_int_t *m, const armpl_int_t *n,
            const armpl_singlecomplex_t *alpha,
            const armpl_singlecomplex_t *a, const armpl_int_t *lda,
            const armpl_singlecomplex_t *x, const armpl_int_t *incx,
            const armpl_singlecomplex_t *beta, armpl_singlecomplex_t *y,
            const armpl_int_t *incy, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' y := alpha*A*x + beta*y.

TRANS = 'T' or 't' y := alpha*A^T *x + beta*y.

TRANS = 'C' or 'c' y := alpha*A^H *x + beta*y.

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of the matrix A. M must be at least zero.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix A. N must be at least zero.

ALPHA Input parameter.

ALPHA is COMPLEX

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). Before entry, the leading m by n part of the array A must contain the matrix of coefficients.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, m).

X Input parameter.

X is COMPLEX

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)) when TRANS = 'N' or 'n' and at least (1 + (m - 1)*abs(INCX)) otherwise. Before entry, the incremented array X must contain the vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

BETA Input parameter.

BETA is COMPLEX

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input.

Y Input and output parameter.

Y is COMPLEX

Y is an array, dimension at least. (1 + (m - 1)*abs(INCY)) when TRANS = 'N' or 'n' and at least (1 + (n - 1)*abs(INCY)) otherwise. Before entry with BETA non-zero, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

Related Information

For this routine in other precisions, please see *dgemv*, *sgemv* and *zgemv*. It also exists with a native C interface as *cblas_cgemv*.

3.3.3 cgerc

cgerc performs the rank 1 operation

```
A := alpha*x*y**H + A,
```

where alpha is a scalar, x is an m element vector, y is an n element vector and A is an m by n matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgerc(M, N, ALPHA, X, INCX, Y, INCY, A, LDA)
```

C specification:

```
#include "armpl.h"

void cgerc_(const armpl_int_t *m, const armpl_int_t *n,
            const armpl_singlecomplex_t *alpha,
            const armpl_singlecomplex_t *x, const armpl_int_t *incx,
            const armpl_singlecomplex_t *y, const armpl_int_t *incy,
            armpl_singlecomplex_t *a, const armpl_int_t *lda);
```

Parameters

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of the matrix A. M must be at least zero.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix A. N must be at least zero.

ALPHA Input parameter.

ALPHA is COMPLEX

On entry, ALPHA specifies the scalar alpha.

X Input parameter.

X is COMPLEX

X is an array, dimension at least. (1 + (m - 1)*abs(INCX)). Before entry, the incremented array X must contain the m element vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Y Input parameter.

Y is COMPLEX

Y is an array, dimension at least. (1 + (n - 1)*abs(INCY)). Before entry, the incremented array Y must contain the n element vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). Before entry, the leading m by n part of the array A must contain the matrix of coefficients. On exit, A is overwritten by the updated matrix.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, m).

Related Information

For this routine in other precisions, please see *zgerc*. It also exists with a native C interface as *cblas_cgerc*.

3.3.4 cgeru

cgeru performs the rank 1 operation

```
A := alpha*x*y**T + A,
```

where alpha is a scalar, x is an m element vector, y is an n element vector and A is an m by n matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgeru(M, N, ALPHA, X, INCX, Y, INCY, A, LDA)
```

C specification:

```
#include "armpl.h"

void cgeru_(const armpl_int_t *m, const armpl_int_t *n,
            const armpl_singlecomplex_t *alpha,
            const armpl_singlecomplex_t *x, const armpl_int_t *incx,
            const armpl_singlecomplex_t *y, const armpl_int_t *incy,
            armpl_singlecomplex_t *a, const armpl_int_t *lda);
```

Parameters

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of the matrix A. M must be at least zero.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix A. N must be at least zero.

ALPHA Input parameter.

ALPHA is COMPLEX

On entry, ALPHA specifies the scalar alpha.

X Input parameter.

X is COMPLEX

X is an array, dimension at least. (1 + (m - 1)*abs(INCX)). Before entry, the incremented array X must contain the m element vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Y Input parameter.

Y is COMPLEX

Y is an array, dimension at least. (1 + (n - 1)*abs(INCY)). Before entry, the incremented array Y must contain the n element vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). Before entry, the leading m by n part of the array A must contain the matrix of coefficients. On exit, A is overwritten by the updated matrix.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, m).

Related Information

For this routine in other precisions, please see *zgeru*. It also exists with a native C interface as *cblas_cgeru*.

3.3.5 chbmv

chbmv performs the matrix-vector operation

```
y := alpha*A*x + beta*y,
```

where alpha and beta are scalars, x and y are n element vectors and A is an n by n hermitian band matrix, with k super-diagonals.

Syntax

Fortran specification:

```
use armpl_library

subroutine chbmv(UPLO, N, K, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
```

C specification:

```
#include "armpl.h"

void chbmv_(const char *uplo, const armpl_int_t *n, const armpl_int_t *k,
            const armpl_singlecomplex_t *alpha,
            const armpl_singlecomplex_t *a, const armpl_int_t *lda,
            const armpl_singlecomplex_t *x, const armpl_int_t *incx,
            const armpl_singlecomplex_t *beta, armpl_singlecomplex_t *y,
            const armpl_int_t *incy, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the band matrix A is being supplied as follows:

UPLO = 'U' or 'u' The upper triangular part of A is being supplied.

UPLO = 'L' or 'l' The lower triangular part of A is being supplied.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

K Input parameter.

K is INTEGER

On entry, K specifies the number of super-diagonals of the matrix A. K must satisfy 0 .le. K.

ALPHA Input parameter.

ALPHA is COMPLEX

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). Before entry with UPLO = 'U' or 'u', the leading (k + 1) by n part of the array A must contain the upper triangular band part of the hermitian matrix, supplied column by column, with the leading diagonal of the matrix in row (k + 1) of the array, the first super-diagonal starting at position 2 in row k, and so on. The top left k by k triangle of the array A is not referenced. The following program segment will transfer the upper triangular part of a hermitian band matrix from conventional full matrix storage to band storage:

```
   DO 20, J = 1, N
      M = ``K`` + 1 - J
      DO 10, I = MAX( 1, J - ``K`` ), J
         A( M + I, J ) = matrix( I, J )
10    CONTINUE
20 CONTINUE
```

Before entry with UPLO = 'L' or 'l', the leading (k + 1) by n part of the array A must contain the lower triangular band part of the hermitian matrix, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k by k triangle of the array A is not referenced. The following program segment will transfer the lower triangular part of a hermitian band matrix from conventional full matrix storage to band storage:

```
   DO 20, J = 1, N
      M = 1 - J
      DO 10, I = J, MIN( ``N``, J + ``K`` )
         A( M + I, J ) = matrix( I, J )
10    CONTINUE
20 CONTINUE
```

Note that the imaginary parts of the diagonal elements need not be set and are assumed to be zero.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least (k + 1).

X Input parameter.

X is COMPLEX

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

BETA Input parameter.

BETA is COMPLEX

On entry, BETA specifies the scalar beta.

Y Input and output parameter.

Y is COMPLEX

Y is an array, dimension at least. (1 + (n - 1)*abs(INCY)). Before entry, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

Related Information

For this routine in other precisions, please see *zhbmv*. It also exists with a native C interface as *cblas_chbmv*.

3.3.6 chemv

chemv performs the matrix-vector operation

```
y := alpha*A*x + beta*y,
```

where alpha and beta are scalars, x and y are n element vectors and A is an n by n hermitian matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine chemv(UPLO, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
```

C specification:

```
#include "armpl.h"

void chemv_(const char *uplo, const armpl_int_t *n,
            const armpl_singlecomplex_t *alpha,
            const armpl_singlecomplex_t *a, const armpl_int_t *lda,
            const armpl_singlecomplex_t *x, const armpl_int_t *incx,
            const armpl_singlecomplex_t *beta, armpl_singlecomplex_t *y,
            const armpl_int_t *incy, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the array A is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of A is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of A is to be referenced.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

ALPHA Input parameter.

ALPHA is COMPLEX

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the hermitian matrix and the strictly lower triangular part of A is not referenced. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the hermitian matrix and the strictly upper triangular part of A is not referenced. Note that the imaginary parts of the diagonal elements need not be set and are assumed to be zero.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, n).

X Input parameter.

X is COMPLEX

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

BETA Input parameter.

BETA is COMPLEX

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input.

Y Input and output parameter.

Y is COMPLEX

Y is an array, dimension at least. (1 + (n - 1)*abs(INCY)). Before entry, the incremented array Y must contain the n element vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

Related Information

For this routine in other precisions, please see *zhemv*. It also exists with a native C interface as *cblas_chemv*.

3.3.7 cher

`cher` performs the hermitian rank 1 operation

```
A := alpha*x*x**H + A,
```

where alpha is a real scalar, x is an n element vector and A is an n by n hermitian matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine cher(UPLO, N, ALPHA, X, INCX, A, LDA)
```

C specification:

```
#include "armpl.h"

void cher_(const char *uplo, const armpl_int_t *n, const float *alpha,
           const armpl_singlecomplex_t *x, const armpl_int_t *incx,
           armpl_singlecomplex_t *a, const armpl_int_t *lda, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the array A is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of A is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of A is to be referenced.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

ALPHA Input parameter.

ALPHA is REAL

On entry, ALPHA specifies the scalar alpha.

X Input parameter.

X is COMPLEX

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the hermitian matrix and the strictly lower triangular part of A is not referenced. On exit, the upper triangular part of the array A is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the hermitian matrix and the strictly upper triangular part of A is not referenced. On exit, the lower triangular part of the array A is overwritten by the lower triangular part of the updated matrix. Note that the imaginary parts of the diagonal elements need not be set, they are assumed to be zero, and on exit they are set to zero.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, n).

Related Information

For this routine in other precisions, please see *zher*. It also exists with a native C interface as *cblas_cher*.

3.3.8 cher2

cher2 performs the hermitian rank 2 operation

```
A := alpha*x*y**H + conjg( alpha )*y*x**H + A,
```

where alpha is a scalar, x and y are n element vectors and A is an n by n hermitian matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine cher2(UPLO, N, ALPHA, X, INCX, Y, INCY, A, LDA)
```

C specification:

```
#include "armpl.h"

void cher2_(const char *uplo, const armpl_int_t *n,
            const armpl_singlecomplex_t *alpha,
            const armpl_singlecomplex_t *x, const armpl_int_t *incx,
            const armpl_singlecomplex_t *y, const armpl_int_t *incy,
            armpl_singlecomplex_t *a, const armpl_int_t *lda, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the array A is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of A is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of A is to be referenced.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

ALPHA Input parameter.

ALPHA is COMPLEX

On entry, ALPHA specifies the scalar alpha.

X Input parameter.

X is COMPLEX

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Y Input parameter.

Y is COMPLEX

Y is an array, dimension at least. (1 + (n - 1)*abs(INCY)). Before entry, the incremented array Y must contain the n element vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the hermitian matrix and the strictly lower triangular part of A is not referenced. On exit, the upper triangular part of the array A is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the hermitian matrix and the strictly upper triangular part of A is not referenced. On exit, the lower triangular part of the array A is overwritten by the lower triangular part of the updated matrix. Note that the imaginary parts of the diagonal elements need not be set, they are assumed to be zero, and on exit they are set to zero.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, n).

Related Information

For this routine in other precisions, please see *zher2*. It also exists with a native C interface as *cblas_cher2*.

3.3.9 chpmv

chpmv performs the matrix-vector operation

```
y := alpha*A*x + beta*y,
```

where alpha and beta are scalars, x and y are n element vectors and A is an n by n hermitian matrix, supplied in packed form.

Syntax

Fortran specification:

```
use armpl_library

subroutine chpmv(UPLO, N, ALPHA, AP, X, INCX, BETA, Y, INCY)
```

C specification:

```
#include "armpl.h"

void chpmv_(const char *uplo, const armpl_int_t *n,
            const armpl_singlecomplex_t *alpha,
            const armpl_singlecomplex_t *a, const armpl_singlecomplex_t *x,
            const armpl_int_t *incx, const armpl_singlecomplex_t *beta,
            armpl_singlecomplex_t *y, const armpl_int_t *incy, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array AP as follows:

UPLO = 'U' or 'u' The upper triangular part of A is supplied in AP.

UPLO = 'L' or 'l' The lower triangular part of A is supplied in AP.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

ALPHA Input parameter.

ALPHA is COMPLEX

On entry, ALPHA specifies the scalar alpha.

AP Input parameter.

AP is COMPLEX

AP is an array, dimension at least. ((n*(n + 1))/2). Before entry with UPLO = 'U' or 'u', the array AP must contain the upper triangular part of the hermitian matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(1, 2) and a(2, 2) respectively, and so on. Before entry with UPLO = 'L' or 'l', the array AP must contain the lower triangular part of the hermitian matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(2, 1) and a(3, 1) respectively, and so on. Note that the imaginary parts of the diagonal elements need not be set and are assumed to be zero.

X Input parameter.

X is COMPLEX

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

BETA Input parameter.

BETA is COMPLEX

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input.

Y Input and output parameter.

Y is COMPLEX

Y is an array, dimension at least. (1 + (n - 1)*abs(INCY)). Before entry, the incremented array Y must contain the n element vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

Related Information

For this routine in other precisions, please see *zhpmv*. It also exists with a native C interface as *cblas_chpmv*.

3.3.10 chpr

chpr performs the hermitian rank 1 operation

```
A := alpha*x*x**H + A,
```

where alpha is a real scalar, x is an n element vector and A is an n by n hermitian matrix, supplied in packed form.

Syntax

Fortran specification:

```
use armpl_library

subroutine chpr(UPLO, N, ALPHA, X, INCX, AP)
```

C specification:

```
#include "armpl.h"

void chpr_(const char *uplo, const armpl_int_t *n, const float *alpha,
           const armpl_singlecomplex_t *x, const armpl_int_t *incx,
           armpl_singlecomplex_t *a, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array AP as follows:

UPLO = 'U' or 'u' The upper triangular part of A is supplied in AP.

UPLO = 'L' or 'l' The lower triangular part of A is supplied in AP.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

ALPHA Input parameter.

ALPHA is REAL

On entry, ALPHA specifies the scalar alpha.

X Input parameter.

X is COMPLEX

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

AP Input and output parameter.

AP is COMPLEX

AP is an array, dimension at least. ((n*(n + 1))/2). Before entry with UPLO = 'U' or 'u', the array AP must contain the upper triangular part of the hermitian matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(1, 2) and a(2, 2) respectively, and so on. On exit, the array AP is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the array AP must contain the lower triangular part of the hermitian matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(2, 1) and a(3, 1) respectively, and so on. On exit, the array AP is overwritten by the lower triangular part of the updated matrix. Note that the imaginary parts of the diagonal elements need not be set, they are assumed to be zero, and on exit they are set to zero.

Related Information

For this routine in other precisions, please see *zhpr*. It also exists with a native C interface as *cblas_chpr*.

3.3.11 chpr2

chpr2 performs the hermitian rank 2 operation

```
A := alpha*x*y**H + conjg( alpha )*y*x**H + A,
```

where alpha is a scalar, x and y are n element vectors and A is an n by n hermitian matrix, supplied in packed form.

Syntax

Fortran specification:

```
use armpl_library

subroutine chpr2(UPLO, N, ALPHA, X, INCX, Y, INCY, AP)
```

C specification:

```
#include "armpl.h"

void chpr2_(const char *uplo, const armpl_int_t *n,
            const armpl_singlecomplex_t *alpha,
            const armpl_singlecomplex_t *x, const armpl_int_t *incx,
            const armpl_singlecomplex_t *y, const armpl_int_t *incy,
            armpl_singlecomplex_t *a, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array AP as follows:

UPLO = 'U' or 'u' The upper triangular part of A is supplied in AP.

UPLO = 'L' or 'l' The lower triangular part of A is supplied in AP.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

ALPHA Input parameter.

ALPHA is COMPLEX

On entry, ALPHA specifies the scalar alpha.

X Input parameter.

X is COMPLEX

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Y Input parameter.

Y is COMPLEX

Y is an array, dimension at least. (1 + (n - 1)*abs(INCY)). Before entry, the incremented array Y must contain the n element vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

AP Input and output parameter.

AP is COMPLEX

AP is an array, dimension at least. ((n*(n + 1))/2). Before entry with UPLO = 'U' or 'u', the array AP must contain the upper triangular part of the hermitian matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(1, 2) and a(2, 2) respectively, and so on. On exit, the array AP is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the array AP must contain the lower triangular part of the hermitian matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(2, 1) and a(3, 1)

respectively, and so on. On exit, the array AP is overwritten by the lower triangular part of the updated matrix. Note that the imaginary parts of the diagonal elements need not be set, they are assumed to be zero, and on exit they are set to zero.

Related Information

For this routine in other precisions, please see *zhpr2*. It also exists with a native C interface as *cblas_chpr2*.

3.3.12 cspmv

cspmv performs the matrix-vector operation

```
y := alpha*A*x + beta*y,
```

where alpha and beta are scalars, x and y are n element vectors and A is an n by n symmetric matrix, supplied in packed form.

Syntax

Fortran specification:

```
use armpl_library

subroutine cspmv(UPLO, N, ALPHA, AP, X, INCX, BETA, Y, INCY)
```

C specification:

```
#include "armpl.h"

void cspmv_(const char *uplo, const armpl_int_t *n,
            const armpl_singlecomplex_t *alpha,
            const armpl_singlecomplex_t *ap, const armpl_singlecomplex_t *x,
            const armpl_int_t *incx, const armpl_singlecomplex_t *beta,
            armpl_singlecomplex_t *y, const armpl_int_t *incy, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array AP as follows:

UPLO = 'U' or 'u' The upper triangular part of A is supplied in AP.

UPLO = 'L' or 'l' The lower triangular part of A is supplied in AP.

Unchanged on exit.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero. Unchanged on exit.

ALPHA Input parameter.

ALPHA is COMPLEX

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

AP Input parameter.

AP is COMPLEX

AP is an array, dimension at least. ((N*(N + 1))/2). Before entry, with UPLO = 'U' or 'u', the array AP must contain the upper triangular part of the symmetric matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(1, 2) and a(2, 2) respectively, and so on. Before entry, with UPLO = 'L' or 'l', the array AP must contain the lower triangular part of the symmetric matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(2, 1) and a(3, 1) respectively, and so on. Unchanged on exit.

X Input parameter.

X is COMPLEX

X is an array, dimension at least. (1 + (N - 1)*abs(INCX)). Before entry, the incremented array X must contain the N- element vector x. Unchanged on exit.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

BETA Input parameter.

BETA is COMPLEX

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input. Unchanged on exit.

Y Input and output parameter.

Y is COMPLEX

Y is an array, dimension at least. (1 + (N - 1)*abs(INCY)). Before entry, the incremented array Y must contain the n element vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero. Unchanged on exit.

Related Information

For this routine in other precisions, please see *dspmv*, *sspmv* and *zspmv*.

3.3.13 cspr

cspr performs the symmetric rank 1 operation

```
A := alpha*x*x**H + A,
```

where alpha is a complex scalar, x is an n element vector and A is an n by n symmetric matrix, supplied in packed form.

Syntax

Fortran specification:

```
use armpl_library

subroutine cspr(UPLO, N, ALPHA, X, INCX, AP)
```

C specification:

```
#include "armpl.h"

void cspr_(const char *uplo, const armpl_int_t *n,
           const armpl_singlecomplex_t *alpha, const armpl_singlecomplex_t *x,
           const armpl_int_t *incx, armpl_singlecomplex_t *ap, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array AP as follows:

UPLO = 'U' or 'u' The upper triangular part of A is supplied in AP.

UPLO = 'L' or 'l' The lower triangular part of A is supplied in AP.

Unchanged on exit.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero. Unchanged on exit.

ALPHA Input parameter.

ALPHA is COMPLEX

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

X Input parameter.

X is COMPLEX

X is an array, dimension at least. (1 + (N - 1)*abs(INCX)). Before entry, the incremented array X must contain the N- element vector x. Unchanged on exit.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

AP Input and output parameter.

AP is COMPLEX

AP is an array, dimension at least. ((N*(N + 1))/2). Before entry, with UPLO = 'U' or 'u', the array AP must contain the upper triangular part of the symmetric matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(1, 2) and a(2, 2) respectively, and so on. On exit, the array AP is overwritten by the upper triangular part of the updated matrix. Before entry, with UPLO = 'L' or 'l', the array AP must contain the lower triangular part of the symmetric matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(2, 1) and a(3, 1) respectively, and so on. On exit, the array AP is overwritten by the lower triangular part of the updated matrix. Note that the imaginary parts of the diagonal elements need not be set, they are assumed to be zero, and on exit they are set to zero.

Related Information

For this routine in other precisions, please see *dspr*, *sspr* and *zspr*.

3.3.14 csymv

csymv performs the matrix-vector operation

```
y := alpha*A*x + beta*y,
```

where alpha and beta are scalars, x and y are n element vectors and A is an n by n symmetric matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine csymv(UPLO, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
```

C specification:

```
#include "armpl.h"

void csymv_(const char *uplo, const armpl_int_t *n,
            const armpl_singlecomplex_t *alpha,
            const armpl_singlecomplex_t *a, const armpl_int_t *lda,
            const armpl_singlecomplex_t *x, const armpl_int_t *incx,
            const armpl_singlecomplex_t *beta, armpl_singlecomplex_t *y,
            const armpl_int_t *incy, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the array A is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of A is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of A is to be referenced.

Unchanged on exit.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero. Unchanged on exit.

ALPHA Input parameter.

ALPHA is COMPLEX

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). Before entry, with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not referenced. Before entry, with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced. Unchanged on exit.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, N). Unchanged on exit.

X Input parameter.

X is COMPLEX

X is an array, dimension at least. (1 + (N - 1)*abs(INCX)). Before entry, the incremented array X must contain the N- element vector x. Unchanged on exit.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

BETA Input parameter.

BETA is COMPLEX

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input. Unchanged on exit.

Y Input and output parameter.

Y is COMPLEX

Y is an array, dimension at least. (1 + (N - 1)*abs(INCY)). Before entry, the incremented array Y must contain the n element vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero. Unchanged on exit.

Related Information

For this routine in other precisions, please see *dsymv*, *ssymv* and *zsymv*.

3.3.15 csyr

csyr performs the symmetric rank 1 operation

```
A := alpha*x*x**H + A,
```

where alpha is a complex scalar, x is an n element vector and A is an n by n symmetric matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine csyr(UPLO, N, ALPHA, X, INCX, A, LDA)
```

C specification:

```
#include "armpl.h"

void csyr_(const char *uplo, const armpl_int_t *n,
           const armpl_singlecomplex_t *alpha, const armpl_singlecomplex_t *x,
           const armpl_int_t *incx, armpl_singlecomplex_t *a,
           const armpl_int_t *lda, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the array A is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of A is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of A is to be referenced.

Unchanged on exit.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero. Unchanged on exit.

ALPHA Input parameter.

ALPHA is COMPLEX

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

X Input parameter.

X is COMPLEX

X is an array, dimension at least. (1 + (N - 1)*abs(INCX)). Before entry, the incremented array X must contain the N- element vector x. Unchanged on exit.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). Before entry, with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not referenced. On exit, the upper triangular part of the array A is overwritten by the upper triangular part of the updated matrix. Before entry, with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced. On exit, the lower triangular part of the array A is overwritten by the lower triangular part of the updated matrix.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, N). Unchanged on exit.

Related Information

For this routine in other precisions, please see *dsyr*, *ssyr* and *zsyr*. It also exists with a native C interface as *LAPACKE_csyr*.

3.3.16 ctbmv

ctbmv performs one of the matrix-vector operations

```
x := A*x,   or   x := A**T*x,   or   x := A**H*x,
```

where x is an n element vector and A is an n by n unit, or non-unit, upper or lower triangular band matrix, with (k + 1) diagonals.

Syntax

Fortran specification:

```
use armpl_library

subroutine ctbmv(UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX)
```

C specification:

```
#include "armpl.h"

void ctbmv_(const char *uplo, const char *trans, const char *diag,
            const armpl_int_t *n, const armpl_int_t *k,
            const armpl_singlecomplex_t *a, const armpl_int_t *lda,
            armpl_singlecomplex_t *x, const armpl_int_t *incx, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' x := A*x.

TRANS = 'T' or 't' x := A^T *x.

TRANS = 'C' or 'c' x := A^H *x.

DIAG Input parameter.

DIAG is CHARACTER*1

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

K Input parameter.

K is INTEGER

On entry with UPLO = 'U' or 'u', K specifies the number of super-diagonals of the matrix A. On entry with UPLO = 'L' or 'l', K specifies the number of sub-diagonals of the matrix A. K must satisfy 0 .le. K.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N).. Before entry with UPLO = 'U' or 'u', the leading (k + 1) by n part of the array A must contain the upper triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row (k + 1) of the array, the first super-diagonal starting at position 2 in row k, and so on. The top left k by k triangle of the array A is not referenced. The following program segment will transfer an upper triangular band matrix from conventional full matrix storage to band storage:

DO 20, J = 1, N M = K + 1 - J DO 10, I = MAX(1, J - K), J A(M + I, J) = matrix(I, J) 10 CONTINUE 20 CONTINUE

Before entry with UPLO = 'L' or 'l', the leading (k + 1) by n part of the array A must contain the lower triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k by k triangle of the array A is not referenced. The following program segment will transfer a lower triangular band matrix from conventional full matrix storage to band storage:

DO 20, J = 1, N M = 1 - J DO 10, I = J, MIN(N, J + K) A(M + I, J) = matrix(I, J) 10 CONTINUE 20 CONTINUE

Note that when DIAG = 'U' or 'u' the elements of the array A corresponding to the diagonal elements of the matrix are not referenced, but are assumed to be unity.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least (k + 1).

X Input and output parameter.

X is COMPLEX

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x. On exit, X is overwritten with the transformed vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Related Information

For this routine in other precisions, please see *dtbmv*, *stbmv* and *ztbmv*. It also exists with a native C interface as *cblas_ctbmv*.

3.3.17 ctbsv

ctbsv solves one of the systems of equations

```
A*x = b,   or   A**T*x = b,   or   A**H*x = b,
```

where b and x are n element vectors and A is an n by n unit, or non-unit, upper or lower triangular band matrix, with (k + 1) diagonals.

No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

Syntax

Fortran specification:

```
use armpl_library

subroutine ctbsv(UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX)
```

C specification:

```
#include "armpl.h"

void ctbsv_(const char *uplo, const char *trans, const char *diag,
            const armpl_int_t *n, const armpl_int_t *k,
            const armpl_singlecomplex_t *a, const armpl_int_t *lda,
            armpl_singlecomplex_t *x, const armpl_int_t *incx, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the equations to be solved as follows:

TRANS = 'N' or 'n' A*x = b.

TRANS = 'T' or 't' A^T *x = b.

TRANS = 'C' or 'c' A^H *x = b.

DIAG Input parameter.

DIAG is CHARACTER*1

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

K Input parameter.

K is INTEGER

On entry with UPLO = 'U' or 'u', K specifies the number of super-diagonals of the matrix A. On entry with UPLO = 'L' or 'l', K specifies the number of sub-diagonals of the matrix A. K must satisfy 0 .le. K.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). Before entry with UPLO = 'U' or 'u', the leading (k + 1) by n part of the array A must contain the upper triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row (k + 1) of the array, the first super-diagonal starting at position 2 in row k, and so on. The top left k by k triangle of the array A is not referenced. The following program segment will transfer an upper triangular band matrix from conventional full matrix storage to band storage:

```
   DO 20, J = 1, N
      M = ``K`` + 1 - J
      DO 10, I = MAX( 1, J - ``K`` ), J
         A( M + I, J ) = matrix( I, J )
10    CONTINUE
20 CONTINUE
```

Before entry with UPLO = 'L' or 'l', the leading (k + 1) by n part of the array A must contain the lower triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k by k triangle of the array A is not referenced. The following program segment will transfer a lower triangular band matrix from conventional full matrix storage to band storage:

```
   DO 20, J = 1, N
      M = 1 - J
      DO 10, I = J, MIN( ``N``, J + ``K`` )
         A( M + I, J ) = matrix( I, J )
10    CONTINUE
20 CONTINUE
```

Note that when DIAG = 'U' or 'u' the elements of the array A corresponding to the diagonal elements of the matrix are not referenced, but are assumed to be unity.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least (k + 1).

X Input and output parameter.

X is COMPLEX

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element right-hand side vector b. On exit, X is overwritten with the solution vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Related Information

For this routine in other precisions, please see *dtbsv*, *stbsv* and *ztbsv*. It also exists with a native C interface as *cblas_ctbsv*.

3.3.18 ctpmv

ctpmv performs one of the matrix-vector operations

```
x := A*x,   or   x := A**T*x,   or   x := A**H*x,
```

where x is an n element vector and A is an n by n unit, or non-unit, upper or lower triangular matrix, supplied in packed form.

Syntax

Fortran specification:

```
use armpl_library

subroutine ctpmv(UPLO, TRANS, DIAG, N, AP, X, INCX)
```

C specification:

```
#include "armpl.h"

void ctpmv_(const char *uplo, const char *trans, const char *diag,
            const armpl_int_t *n, const armpl_singlecomplex_t *ap,
            armpl_singlecomplex_t *x, const armpl_int_t *incx, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' x := A*x.

TRANS = 'T' or 't' x := A^T *x.

TRANS = 'C' or 'c' x := A^H *x.

DIAG Input parameter.

DIAG is CHARACTER*1

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

AP Input parameter.

AP is COMPLEX

AP is an array, dimension at least. ((n*(n + 1))/2). Before entry with UPLO = 'U' or 'u', the array AP must contain the upper triangular matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(1, 2) and a(2, 2) respectively, and so on. Before entry with UPLO = 'L' or 'l', the array AP must contain the lower triangular matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(2, 1) and a(3, 1) respectively, and so on. Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced, but are assumed to be unity.

X Input and output parameter.

X is COMPLEX

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x. On exit, X is overwritten with the transformed vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Related Information

For this routine in other precisions, please see *dtpmv*, *stpmv* and *ztpmv*. It also exists with a native C interface as *cblas_ctpmv*.

3.3.19 ctpsv

ctpsv solves one of the systems of equations

```
A*x = b,   or   A**T*x = b,   or   A**H*x = b,
```

where b and x are n element vectors and A is an n by n unit, or non-unit, upper or lower triangular matrix, supplied in packed form.

No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

Syntax

Fortran specification:

```
use armpl_library

subroutine ctpsv(UPLO, TRANS, DIAG, N, AP, X, INCX)
```

C specification:

```
#include "armpl.h"

void ctpsv_(const char *uplo, const char *trans, const char *diag,
            const armpl_int_t *n, const armpl_singlecomplex_t *ap,
            armpl_singlecomplex_t *x, const armpl_int_t *incx, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the equations to be solved as follows:

TRANS = 'N' or 'n' A*x = b.

TRANS = 'T' or 't' A^T *x = b.

TRANS = 'C' or 'c' A^H *x = b.

DIAG Input parameter.

DIAG is CHARACTER*1

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

AP Input parameter.

AP is COMPLEX

AP is an array, dimension at least. ((n*(n + 1))/2). Before entry with UPLO = 'U' or 'u', the array AP must contain the upper triangular matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(1, 2) and a(2, 2) respectively, and so on. Before entry with UPLO = 'L' or 'l', the array AP must contain the lower triangular matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(2, 1) and a(3, 1) respectively, and so on. Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced, but are assumed to be unity.

X Input and output parameter.

X is COMPLEX

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element right-hand side vector b. On exit, X is overwritten with the solution vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Related Information

For this routine in other precisions, please see *dtpsv*, *stpsv* and *ztpsv*. It also exists with a native C interface as *cblas_ctpsv*.

3.3.20 ctrmv

ctrmv performs one of the matrix-vector operations

```
x := A*x,   or   x := A**T*x,   or   x := A**H*x,
```

where x is an n element vector and A is an n by n unit, or non-unit, upper or lower triangular matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine ctrmv(UPLO, TRANS, DIAG, N, A, LDA, X, INCX)
```

C specification:

```
#include "armpl.h"

void ctrmv_(const char *uplo, const char *trans, const char *diag,
            const armpl_int_t *n, const armpl_singlecomplex_t *a,
            const armpl_int_t *lda, armpl_singlecomplex_t *x,
            const armpl_int_t *incx, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' x := A*x.

TRANS = 'T' or 't' x := A^T *x.

TRANS = 'C' or 'c' x := A^H *x.

DIAG Input parameter.

DIAG is CHARACTER*1

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N).. Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular matrix and the strictly lower triangular part of A is not referenced. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular matrix and the strictly upper triangular part of A is not referenced. Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, n).

X Input and output parameter.

X is COMPLEX

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x. On exit, X is overwritten with the transformed vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Related Information

For this routine in other precisions, please see *dtrmv*, *strmv* and *ztrmv*. It also exists with a native C interface as *cblas_ctrmv*.

3.3.21 ctrsv

ctrsv solves one of the systems of equations

```
A*x = b,   or   A**T*x = b,   or   A**H*x = b,
```

where b and x are n element vectors and A is an n by n unit, or non-unit, upper or lower triangular matrix.

No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

Syntax

Fortran specification:

```
use armpl_library

subroutine ctrsv(UPLO, TRANS, DIAG, N, A, LDA, X, INCX)
```

C specification:

```
#include "armpl.h"

void ctrsv_(const char *uplo, const char *trans, const char *diag,
            const armpl_int_t *n, const armpl_singlecomplex_t *a,
            const armpl_int_t *lda, armpl_singlecomplex_t *x,
            const armpl_int_t *incx, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the equations to be solved as follows:

TRANS = 'N' or 'n' A*x = b.

TRANS = 'T' or 't' A^T *x = b.

TRANS = 'C' or 'c' A^H *x = b.

DIAG Input parameter.

DIAG is CHARACTER*1

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular matrix and the strictly lower triangular part of A is not referenced. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular matrix and the strictly upper triangular part of A is not referenced. Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, n).

X Input and output parameter.

X is COMPLEX

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element right-hand side vector b. On exit, X is overwritten with the solution vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Related Information

For this routine in other precisions, please see *dtrsv*, *strsv* and *ztrsv*. It also exists with a native C interface as *cblas_ctrsv*.

3.3.22 dgbmv

dgbmv performs one of the matrix-vector operations

```
y := alpha*A*x + beta*y,   or   y := alpha*A**T*x + beta*y,
```

where alpha and beta are scalars, x and y are vectors and A is an m by n band matrix, with kl sub-diagonals and ku super-diagonals.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgbmv(TRANS, M, N, KL, KU, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
```

C specification:

```
#include "armpl.h"

void dgbmv_(const char *trans, const armpl_int_t *m, const armpl_int_t *n,
            const armpl_int_t *kl, const armpl_int_t *ku, const double *alpha,
            const double *a, const armpl_int_t *lda, const double *x,
            const armpl_int_t *incx, const double *beta, double *y,
            const armpl_int_t *incy, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' y := alpha*A*x + beta*y.

TRANS = 'T' or 't' y := alpha*A^T *x + beta*y.

TRANS = 'C' or 'c' y := alpha*A^T *x + beta*y.

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of the matrix A. M must be at least zero.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix A. N must be at least zero.

KL Input parameter.

KL is INTEGER

On entry, KL specifies the number of sub-diagonals of the matrix A. KL must satisfy 0 .le. KL.

KU Input parameter.

KU is INTEGER

On entry, KU specifies the number of super-diagonals of the matrix A. KU must satisfy 0 .le. KU.

ALPHA Input parameter.

ALPHA is DOUBLE PRECISION.

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). Before entry, the leading (kl + ku + 1) by n part of the array A must contain the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row (ku + 1) of the array, the first super-diagonal starting at position 2 in row ku, the first sub-diagonal starting at position 1 in row (ku + 2), and so on. Elements in the array A that do not correspond to elements in the band matrix (such as the top left ku by ku triangle) are not referenced. The following program segment will transfer a band matrix from conventional full matrix storage to band storage:

```
   DO 20, J = 1, N
      K = ``KU`` + 1 - J
      DO 10, I = MAX( 1, J - ``KU`` ), MIN( ``M``, J + ``KL`` )
         A( K + I, J ) = matrix( I, J )
10    CONTINUE
20 CONTINUE
```

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least (kl + ku + 1).

X Input parameter.

X is DOUBLE PRECISION

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)) when TRANS = 'N' or 'n' and at least (1 + (m - 1)*abs(INCX)) otherwise. Before entry, the incremented array X must contain the vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

BETA Input parameter.

BETA is DOUBLE PRECISION.

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input.

Y Input and output parameter.

Y is DOUBLE PRECISION

Y is an array, dimension at least. (1 + (m - 1)*abs(INCY)) when TRANS = 'N' or 'n' and at least (1 + (n - 1)*abs(INCY)) otherwise. Before entry, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

Related Information

For this routine in other precisions, please see *cgbmv*, *sgbmv* and *zgbmv*. It also exists with a native C interface as *cblas_dgbmv*.

3.3.23 dgemv

dgemv performs one of the matrix-vector operations

```
y := alpha*A*x + beta*y,   or   y := alpha*A**T*x + beta*y,
```

where alpha and beta are scalars, x and y are vectors and A is an m by n matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgemv(TRANS, M, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
```

C specification:

```
#include "armpl.h"

void dgemv_(const char *trans, const armpl_int_t *m, const armpl_int_t *n,
            const double *alpha, const double *a, const armpl_int_t *lda,
            const double *x, const armpl_int_t *incx, const double *beta,
            double *y, const armpl_int_t *incy, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' y := alpha*A*x + beta*y.

TRANS = 'T' or 't' y := alpha*A^T *x + beta*y.

TRANS = 'C' or 'c' y := alpha*A^T *x + beta*y.

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of the matrix A. M must be at least zero.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix A. N must be at least zero.

ALPHA Input parameter.

ALPHA is DOUBLE PRECISION.

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). Before entry, the leading m by n part of the array A must contain the matrix of coefficients.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, m).

X Input parameter.

X is DOUBLE PRECISION

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)) when TRANS = 'N' or 'n' and at least (1 + (m - 1)*abs(INCX)) otherwise. Before entry, the incremented array X must contain the vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

BETA Input parameter.

BETA is DOUBLE PRECISION.

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input.

Y Input and output parameter.

Y is DOUBLE PRECISION

Y is an array, dimension at least. (1 + (m - 1)*abs(INCY)) when TRANS = 'N' or 'n' and at least (1 + (n - 1)*abs(INCY)) otherwise. Before entry with BETA non-zero, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

Related Information

For this routine in other precisions, please see *cgemv*, *sgemv* and *zgemv*. It also exists with a native C interface as *cblas_dgemv*.

3.3.24 dger

dger performs the rank 1 operation

```
A := alpha*x*y**T + A,
```

where alpha is a scalar, x is an m element vector, y is an n element vector and A is an m by n matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine dger(M, N, ALPHA, X, INCX, Y, INCY, A, LDA)
```

C specification:

```
#include "armpl.h"

void dger_(const armpl_int_t *m, const armpl_int_t *n, const double *alpha,
           const double *x, const armpl_int_t *incx, const double *y,
           const armpl_int_t *incy, double *a, const armpl_int_t *lda);
```

Parameters

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of the matrix A. M must be at least zero.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix A. N must be at least zero.

ALPHA Input parameter.

ALPHA is DOUBLE PRECISION.

On entry, ALPHA specifies the scalar alpha.

X Input parameter.

X is DOUBLE PRECISION

X is an array, dimension at least. (1 + (m - 1)*abs(INCX)). Before entry, the incremented array X must contain the m element vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Y Input parameter.

Y is DOUBLE PRECISION

Y is an array, dimension at least. (1 + (n - 1)*abs(INCY)). Before entry, the incremented array Y must contain the n element vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). Before entry, the leading m by n part of the array A must contain the matrix of coefficients. On exit, A is overwritten by the updated matrix.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, m).

Related Information

For this routine in other precisions, please see *sger*. It also exists with a native C interface as *cblas_dger*.

3.3.25 dsbmv

dsbmv performs the matrix-vector operation

```
y := alpha*A*x + beta*y,
```

where alpha and beta are scalars, x and y are n element vectors and A is an n by n symmetric band matrix, with k super-diagonals.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsbmv(UPLO, N, K, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
```

C specification:

```
#include "armpl.h"

void dsbmv_(const char *uplo, const armpl_int_t *n, const armpl_int_t *k,
            const double *alpha, const double *a, const armpl_int_t *lda,
            const double *x, const armpl_int_t *incx, const double *beta,
            double *y, const armpl_int_t *incy, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the band matrix A is being supplied as follows:

UPLO = 'U' or 'u' The upper triangular part of A is being supplied.

UPLO = 'L' or 'l' The lower triangular part of A is being supplied.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

K Input parameter.

K is INTEGER

On entry, K specifies the number of super-diagonals of the matrix A. K must satisfy 0 .le. K.

ALPHA Input parameter.

ALPHA is DOUBLE PRECISION.

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). Before entry with UPLO = 'U' or 'u', the leading (k + 1) by n part of the array A must contain the upper triangular band part of the symmetric matrix, supplied column by column, with the leading diagonal of the matrix in row (k + 1) of the array, the first super-diagonal starting at position 2 in row k, and so on. The top left k by k triangle of the array A is not referenced. The following program segment will transfer the upper triangular part of a symmetric band matrix from conventional full matrix storage to band storage:

```
   DO 20, J = 1, N
      M = ``K`` + 1 - J
      DO 10, I = MAX( 1, J - ``K`` ), J
         A( M + I, J ) = matrix( I, J )
10    CONTINUE
20 CONTINUE
```

Before entry with UPLO = 'L' or 'l', the leading (k + 1) by n part of the array A must contain the lower triangular band part of the symmetric matrix, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k by k triangle of the array A is not referenced. The following program segment will transfer the lower triangular part of a symmetric band matrix from conventional full matrix storage to band storage:

```
   DO 20, J = 1, N
      M = 1 - J
      DO 10, I = J, MIN( ``N``, J + ``K`` )
         A( M + I, J ) = matrix( I, J )
10    CONTINUE
20 CONTINUE
```

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least (k + 1).

X Input parameter.

X is DOUBLE PRECISION

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

BETA Input parameter.

BETA is DOUBLE PRECISION.

On entry, BETA specifies the scalar beta.

Y Input and output parameter.

Y is DOUBLE PRECISION

Y is an array, dimension at least. (1 + (n - 1)*abs(INCY)). Before entry, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

Related Information

For this routine in other precisions, please see *ssbmv*. It also exists with a native C interface as *cblas_dsbmv*.

3.3.26 dspmv

dspmv performs the matrix-vector operation

```
y := alpha*A*x + beta*y,
```

where alpha and beta are scalars, x and y are n element vectors and A is an n by n symmetric matrix, supplied in packed form.

Syntax

Fortran specification:

```
use armpl_library

subroutine dspmv(UPLO, N, ALPHA, AP, X, INCX, BETA, Y, INCY)
```

C specification:

```
#include "armpl.h"

void dspmv_(const char *uplo, const armpl_int_t *n, const double *alpha,
            const double *ap, const double *x, const armpl_int_t *incx,
            const double *beta, double *y, const armpl_int_t *incy, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array AP as follows:

UPLO = 'U' or 'u' The upper triangular part of A is supplied in AP.

UPLO = 'L' or 'l' The lower triangular part of A is supplied in AP.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

ALPHA Input parameter.

ALPHA is DOUBLE PRECISION.

On entry, ALPHA specifies the scalar alpha.

AP Input parameter.

AP is DOUBLE PRECISION

AP is an array, dimension at least. ((n*(n + 1))/2). Before entry with UPLO = 'U' or 'u', the array AP must contain the upper triangular part of the symmetric matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(1, 2) and a(2, 2) respectively, and so on. Before entry with UPLO = 'L' or 'l', the array AP must contain the lower triangular part of the symmetric

matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(2, 1) and a(3, 1) respectively, and so on.

X Input parameter.

X is DOUBLE PRECISION

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

BETA Input parameter.

BETA is DOUBLE PRECISION.

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input.

Y Input and output parameter.

Y is DOUBLE PRECISION

Y is an array, dimension at least. (1 + (n - 1)*abs(INCY)). Before entry, the incremented array Y must contain the n element vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

Related Information

For this routine in other precisions, please see *cspmv*, *sspmv* and *zspmv*. It also exists with a native C interface as *cblas_dspmv*.

3.3.27 dspr

dspr performs the symmetric rank 1 operation

```
A := alpha*x*x**T + A,
```

where alpha is a real scalar, x is an n element vector and A is an n by n symmetric matrix, supplied in packed form.

Syntax

Fortran specification:

```
use armpl_library

subroutine dspr(UPLO, N, ALPHA, X, INCX, AP)
```

C specification:

```
#include "armpl.h"

void dspr_(const char *uplo, const armpl_int_t *n, const double *alpha,
           const double *x, const armpl_int_t *incx, double *ap, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array AP as follows:

UPLO = 'U' or 'u' The upper triangular part of A is supplied in AP.

UPLO = 'L' or 'l' The lower triangular part of A is supplied in AP.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

ALPHA Input parameter.

ALPHA is DOUBLE PRECISION.

On entry, ALPHA specifies the scalar alpha.

X Input parameter.

X is DOUBLE PRECISION

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

AP Input and output parameter.

AP is DOUBLE PRECISION

AP is an array, dimension at least. ((n*(n + 1))/2). Before entry with UPLO = 'U' or 'u', the array AP must contain the upper triangular part of the symmetric matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(1, 2) and a(2, 2) respectively, and so on. On exit, the array AP is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the array AP must contain the lower triangular part of the symmetric matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(2, 1) and a(3, 1) respectively, and so on. On exit, the array AP is overwritten by the lower triangular part of the updated matrix.

Related Information

For this routine in other precisions, please see *cspr*, *sspr* and *zspr*. It also exists with a native C interface as *cblas_dspr*.

3.3.28 dspr2

dspr2 performs the symmetric rank 2 operation

```
A := alpha*x*y**T + alpha*y*x**T + A,
```

where alpha is a scalar, x and y are n element vectors and A is an n by n symmetric matrix, supplied in packed form.

Syntax

Fortran specification:

```
use armpl_library

subroutine dspr2(UPLO, N, ALPHA, X, INCX, Y, INCY, AP)
```

C specification:

```
#include "armpl.h"

void dspr2_(const char *uplo, const armpl_int_t *n, const double *alpha,
            const double *x, const armpl_int_t *incx, const double *y,
            const armpl_int_t *incy, double *ap, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array AP as follows:

UPLO = 'U' or 'u' The upper triangular part of A is supplied in AP.

UPLO = 'L' or 'l' The lower triangular part of A is supplied in AP.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

ALPHA Input parameter.

ALPHA is DOUBLE PRECISION.

On entry, ALPHA specifies the scalar alpha.

X Input parameter.

X is DOUBLE PRECISION

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Y Input parameter.

Y is DOUBLE PRECISION

Y is an array, dimension at least. (1 + (n - 1)*abs(INCY)). Before entry, the incremented array Y must contain the n element vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

AP Input and output parameter.

AP is DOUBLE PRECISION

AP is an array, dimension at least. ((n*(n + 1))/2). Before entry with UPLO = 'U' or 'u', the array AP must contain the upper triangular part of the symmetric matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(1, 2) and a(2, 2) respectively, and so on. On exit, the array AP is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the array AP must contain the lower triangular part of the symmetric matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(2, 1) and a(3, 1) respectively, and so on. On exit, the array AP is overwritten by the lower triangular part of the updated matrix.

Related Information

For this routine in other precisions, please see *sspr2*. It also exists with a native C interface as *cblas_dspr2*.

3.3.29 dsymv

dsymv performs the matrix-vector operation

```
y := alpha*A*x + beta*y,
```

where alpha and beta are scalars, x and y are n element vectors and A is an n by n symmetric matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsymv(UPLO, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
```

C specification:

```
#include "armpl.h"

void dsymv_(const char *uplo, const armpl_int_t *n, const double *alpha,
            const double *a, const armpl_int_t *lda, const double *x,
            const armpl_int_t *incx, const double *beta, double *y,
            const armpl_int_t *incy, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the array A is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of A is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of A is to be referenced.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

ALPHA Input parameter.

ALPHA is DOUBLE PRECISION.

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not referenced. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, n).

X Input parameter.

X is DOUBLE PRECISION

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

BETA Input parameter.

BETA is DOUBLE PRECISION.

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input.

Y Input and output parameter.

Y is DOUBLE PRECISION

Y is an array, dimension at least. (1 + (n - 1)*abs(INCY)). Before entry, the incremented array Y must contain the n element vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

Related Information

For this routine in other precisions, please see *csymv*, *ssymv* and *zsymv*. It also exists with a native C interface as *cblas_dsymv*.

3.3.30 dsyr

dsyr performs the symmetric rank 1 operation

```
A := alpha*x*x**T + A,
```

where alpha is a real scalar, x is an n element vector and A is an n by n symmetric matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsyr(UPLO, N, ALPHA, X, INCX, A, LDA)
```

C specification:

```
#include "armpl.h"

void dsyr_(const char *uplo, const armpl_int_t *n, const double *alpha,
           const double *x, const armpl_int_t *incx, double *a,
           const armpl_int_t *lda, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the array A is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of A is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of A is to be referenced.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

ALPHA Input parameter.

ALPHA is DOUBLE PRECISION.

On entry, ALPHA specifies the scalar alpha.

X Input parameter.

X is DOUBLE PRECISION

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not referenced. On exit, the upper triangular part of the array A is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced. On exit, the lower triangular part of the array A is overwritten by the lower triangular part of the updated matrix.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, n).

Related Information

For this routine in other precisions, please see *csyr*, *ssyr* and *zsyr*. It also exists with a native C interface as *cblas_dsyr*.

3.3.31 dsyr2

dsyr2 performs the symmetric rank 2 operation

```
A := alpha*x*y**T + alpha*y*x**T + A,
```

where alpha is a scalar, x and y are n element vectors and A is an n by n symmetric matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsyr2(UPLO, N, ALPHA, X, INCX, Y, INCY, A, LDA)
```

C specification:

```
#include "armpl.h"

void dsyr2_(const char *uplo, const armpl_int_t *n, const double *alpha,
            const double *x, const armpl_int_t *incx, const double *y,
            const armpl_int_t *incy, double *a, const armpl_int_t *lda,
            ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the array A is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of A is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of A is to be referenced.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

ALPHA Input parameter.

ALPHA is DOUBLE PRECISION.

On entry, ALPHA specifies the scalar alpha.

X Input parameter.

X is DOUBLE PRECISION

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Y Input parameter.

Y is DOUBLE PRECISION

Y is an array, dimension at least. (1 + (n - 1)*abs(INCY)). Before entry, the incremented array Y must contain the n element vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not referenced. On exit, the upper triangular part of the array A is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced. On exit, the lower triangular part of the array A is overwritten by the lower triangular part of the updated matrix.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, n).

Related Information

For this routine in other precisions, please see *ssyr2*. It also exists with a native C interface as *cblas_dsyr2*.

3.3.32 dtbmv

dtbmv performs one of the matrix-vector operations

```
x := A*x,   or   x := A**T*x,
```

where x is an n element vector and A is an n by n unit, or non-unit, upper or lower triangular band matrix, with (k + 1) diagonals.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtbmv(UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX)
```

C specification:

```
#include "armpl.h"

void dtbmv_(const char *uplo, const char *trans, const char *diag,
            const armpl_int_t *n, const armpl_int_t *k, const double *a,
            const armpl_int_t *lda, double *x, const armpl_int_t *incx,
            ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' x := A*x.

TRANS = 'T' or 't' x := A^T *x.

TRANS = 'C' or 'c' x := A^T *x.

DIAG Input parameter.

DIAG is CHARACTER*1

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

K Input parameter.

K is INTEGER

On entry with UPLO = 'U' or 'u', K specifies the number of super-diagonals of the matrix A. On entry with UPLO = 'L' or 'l', K specifies the number of sub-diagonals of the matrix A. K must satisfy 0 .le. K.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). Before entry with UPLO = 'U' or 'u', the leading (k + 1) by n part of the array A must contain the upper triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row (k + 1) of the array, the first super-diagonal starting at position 2 in row k, and so on. The top left k by k triangle of the array A is not referenced. The following program segment will transfer an upper triangular band matrix from conventional full matrix storage to band storage:

```
   DO 20, J = 1, N
      M = ``K`` + 1 - J
      DO 10, I = MAX( 1, J - ``K`` ), J
         A( M + I, J ) = matrix( I, J )
10    CONTINUE
20 CONTINUE
```

Before entry with UPLO = 'L' or 'l', the leading (k + 1) by n part of the array A must contain the lower triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k by k triangle of the array A is not referenced. The following program segment will transfer a lower triangular band matrix from conventional full matrix storage to band storage:

DO 20, J = 1, N M = 1 - J DO 10, I = J, MIN(N, J + K) A(M + I, J) = matrix(I, J) 10 CONTINUE 20 CONTINUE

Note that when DIAG = 'U' or 'u' the elements of the array A corresponding to the diagonal elements of the matrix are not referenced, but are assumed to be unity.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least (k + 1).

X Input and output parameter.

X is DOUBLE PRECISION

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x. On exit, X is overwritten with the transformed vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Related Information

For this routine in other precisions, please see *ctbmv*, *stbmv* and *ztbmv*. It also exists with a native C interface as *cblas_dtbmv*.

3.3.33 dtbsv

dtbsv solves one of the systems of equations

```
A*x = b,   or   A**T*x = b,
```

where b and x are n element vectors and A is an n by n unit, or non-unit, upper or lower triangular band matrix, with (k + 1) diagonals.

No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtbsv(UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX)
```

C specification:

```
#include "armpl.h"

void dtbsv_(const char *uplo, const char *trans, const char *diag,
            const armpl_int_t *n, const armpl_int_t *k, const double *a,
            const armpl_int_t *lda, double *x, const armpl_int_t *incx,
            ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the equations to be solved as follows:

TRANS = 'N' or 'n' A*x = b.

TRANS = 'T' or 't' A^T *x = b.

TRANS = 'C' or 'c' A^T *x = b.

DIAG Input parameter.

DIAG is CHARACTER*1

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

K Input parameter.

K is INTEGER

On entry with UPLO = 'U' or 'u', K specifies the number of super-diagonals of the matrix A. On entry with UPLO = 'L' or 'l', K specifies the number of sub-diagonals of the matrix A. K must satisfy 0 .le. K.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). Before entry with UPLO = 'U' or 'u', the leading (k + 1) by n part of the array A must contain the upper triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row (k + 1) of the array, the first super-diagonal starting at position 2 in row k, and so on. The top left k by k triangle of the array A is not referenced. The following program segment will transfer an upper triangular band matrix from conventional full matrix storage to band storage:

```
   DO 20, J = 1, N
      M = ``K`` + 1 - J
      DO 10, I = MAX( 1, J - ``K`` ), J
         A( M + I, J ) = matrix( I, J )
10    CONTINUE
20 CONTINUE
```

Before entry with UPLO = 'L' or 'l', the leading (k + 1) by n part of the array A must contain the lower triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k by k triangle of the array A is not referenced. The following program segment will transfer a lower triangular band matrix from conventional full matrix storage to band storage:

```
   DO 20, J = 1, N
      M = 1 - J
      DO 10, I = J, MIN( ``N``, J + ``K`` )
         A( M + I, J ) = matrix( I, J )
10    CONTINUE
20 CONTINUE
```

Note that when DIAG = 'U' or 'u' the elements of the array A corresponding to the diagonal elements of the matrix are not referenced, but are assumed to be unity.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least (k + 1).

X Input and output parameter.

X is DOUBLE PRECISION

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element right-hand side vector b. On exit, X is overwritten with the solution vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Related Information

For this routine in other precisions, please see *ctbsv*, *stbsv* and *ztbsv*. It also exists with a native C interface as *cblas_dtbsv*.

3.3.34 dtpmv

dtpmv performs one of the matrix-vector operations

```
x := A*x,   or   x := A**T*x,
```

where x is an n element vector and A is an n by n unit, or non-unit, upper or lower triangular matrix, supplied in packed form.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtpmv(UPLO, TRANS, DIAG, N, AP, X, INCX)
```

C specification:

```
#include "armpl.h"

void dtpmv_(const char *uplo, const char *trans, const char *diag,
            const armpl_int_t *n, const double *ap, double *x,
            const armpl_int_t *incx, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' x := A*x.

TRANS = 'T' or 't' x := A^T *x.

TRANS = 'C' or 'c' x := A^T *x.

DIAG Input parameter.

DIAG is CHARACTER*1

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

AP Input parameter.

AP is DOUBLE PRECISION

AP is an array, dimension at least. ((n*(n + 1))/2). Before entry with UPLO = 'U' or 'u', the array AP must contain the upper triangular matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(1, 2) and a(2, 2) respectively, and so on. Before entry with UPLO = 'L' or 'l', the array AP must contain the lower triangular matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(2, 1) and a(3, 1) respectively, and so on. Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced, but are assumed to be unity.

X Input and output parameter.

X is DOUBLE PRECISION

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x. On exit, X is overwritten with the transformed vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Related Information

For this routine in other precisions, please see *ctpmv*, *stpmv* and *ztpmv*. It also exists with a native C interface as *cblas_dtpmv*.

3.3.35 dtpsv

dtpsv solves one of the systems of equations

```
A*x = b,   or   A**T*x = b,
```

where b and x are n element vectors and A is an n by n unit, or non-unit, upper or lower triangular matrix, supplied in packed form.

No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtpsv(UPLO, TRANS, DIAG, N, AP, X, INCX)
```

C specification:

```
#include "armpl.h"

void dtpsv_(const char *uplo, const char *trans, const char *diag,
            const armpl_int_t *n, const double *ap, double *x,
            const armpl_int_t *incx, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the equations to be solved as follows:

TRANS = 'N' or 'n' A*x = b.

TRANS = 'T' or 't' A^T *x = b.

TRANS = 'C' or 'c' A^T *x = b.

DIAG Input parameter.

DIAG is CHARACTER*1

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

AP Input parameter.

AP is DOUBLE PRECISION

AP is an array, dimension at least. ((n*(n + 1))/2). Before entry with UPLO = 'U' or 'u', the array AP must contain the upper triangular matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(1, 2) and a(2, 2) respectively, and so on. Before entry with UPLO = 'L' or 'l', the array AP must contain the lower triangular matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(2, 1) and a(3, 1) respectively, and so on. Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced, but are assumed to be unity.

X Input and output parameter.

X is DOUBLE PRECISION

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element right-hand side vector b. On exit, X is overwritten with the solution vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Related Information

For this routine in other precisions, please see *ctpsv*, *stpsv* and *ztpsv*. It also exists with a native C interface as *cblas_dtpsv*.

3.3.36 dtrmv

dtrmv performs one of the matrix-vector operations

```
x := A*x,   or   x := A**T*x,
```

where x is an n element vector and A is an n by n unit, or non-unit, upper or lower triangular matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtrmv(UPLO, TRANS, DIAG, N, A, LDA, X, INCX)
```

C specification:

```
#include "armpl.h"

void dtrmv_(const char *uplo, const char *trans, const char *diag,
            const armpl_int_t *n, const double *a, const armpl_int_t *lda,
            double *x, const armpl_int_t *incx, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' x := A*x.

TRANS = 'T' or 't' x := A^T *x.

TRANS = 'C' or 'c' x := A^T *x.

DIAG Input parameter.

DIAG is CHARACTER*1

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular matrix and the strictly lower triangular part of A is not referenced. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular matrix and the strictly upper triangular part of A is not referenced. Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, n).

X Input and output parameter.

X is DOUBLE PRECISION

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x. On exit, X is overwritten with the transformed vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Related Information

For this routine in other precisions, please see *ctrmv*, *strmv* and *ztrmv*. It also exists with a native C interface as *cblas_dtrmv*.

3.3.37 dtrsv

dtrsv solves one of the systems of equations

```
A*x = b,   or   A**T*x = b,
```

where b and x are n element vectors and A is an n by n unit, or non-unit, upper or lower triangular matrix.

No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtrsv(UPLO, TRANS, DIAG, N, A, LDA, X, INCX)
```

C specification:

```
#include "armpl.h"

void dtrsv_(const char *uplo, const char *trans, const char *diag,
            const armpl_int_t *n, const double *a, const armpl_int_t *lda,
            double *x, const armpl_int_t *incx, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the equations to be solved as follows:

TRANS = 'N' or 'n' A*x = b.

TRANS = 'T' or 't' A^T *x = b.

TRANS = 'C' or 'c' A^T *x = b.

DIAG Input parameter.

DIAG is CHARACTER*1

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular matrix and the strictly lower triangular part of A is not referenced. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular matrix and the strictly upper triangular part of A is not referenced. Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, n).

X Input and output parameter.

X is DOUBLE PRECISION

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element right-hand side vector b. On exit, X is overwritten with the solution vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Level 2 Blas routine.

– Written on 22-October-1986. Jack Dongarra, Argonne National Lab. Jeremy Du Croz, Nag Central Office. Sven Hammarling, Nag Central Office. Richard Hanson, Sandia National Labs.

Related Information

For this routine in other precisions, please see *ctrsv*, *strsv* and *ztrsv*. It also exists with a native C interface as *cblas_dtrsv*.

3.3.38 sgbmv

sgbmv performs one of the matrix-vector operations

```
y := alpha*A*x + beta*y,   or   y := alpha*A**T*x + beta*y,
```

where alpha and beta are scalars, x and y are vectors and A is an m by n band matrix, with kl sub-diagonals and ku super-diagonals.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgbmv(TRANS, M, N, KL, KU, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
```

C specification:

```
#include "armpl.h"

void sgbmv_(const char *trans, const armpl_int_t *m, const armpl_int_t *n,
            const armpl_int_t *kl, const armpl_int_t *ku, const float *alpha,
            const float *a, const armpl_int_t *lda, const float *x,
            const armpl_int_t *incx, const float *beta, float *y,
            const armpl_int_t *incy, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' y := alpha*A*x + beta*y.

TRANS = 'T' or 't' y := alpha*A^T *x + beta*y.

TRANS = 'C' or 'c' y := alpha*A^T *x + beta*y.

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of the matrix A. M must be at least zero.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix A. N must be at least zero.

KL Input parameter.

KL is INTEGER

On entry, KL specifies the number of sub-diagonals of the matrix A. KL must satisfy 0 .le. KL.

KU Input parameter.

KU is INTEGER

On entry, KU specifies the number of super-diagonals of the matrix A. KU must satisfy 0 .le. KU.

ALPHA Input parameter.

ALPHA is REAL

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). Before entry, the leading (kl + ku + 1) by n part of the array A must contain the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row (ku + 1) of the array, the first super-diagonal starting at position 2 in row ku, the first sub-diagonal starting at position 1 in row (ku + 2), and so on. Elements in the array A that do not correspond to elements in the

band matrix (such as the top left ku by ku triangle) are not referenced. The following program segment will transfer a band matrix from conventional full matrix storage to band storage:

```
   DO 20, J = 1, N
      K = ``KU`` + 1 - J
      DO 10, I = MAX( 1, J - ``KU`` ), MIN( ``M``, J + ``KL`` )
         A( K + I, J ) = matrix( I, J )
10    CONTINUE
20 CONTINUE
```

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least (kl + ku + 1).

X Input parameter.

X is REAL

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)) when TRANS = 'N' or 'n' and at least (1 + (m - 1)*abs(INCX)) otherwise. Before entry, the incremented array X must contain the vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

BETA Input parameter.

BETA is REAL

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input.

Y Input and output parameter.

Y is REAL

Y is an array, dimension at least. (1 + (m - 1)*abs(INCY)) when TRANS = 'N' or 'n' and at least (1 + (n - 1)*abs(INCY)) otherwise. Before entry, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

Related Information

For this routine in other precisions, please see *cgbmv*, *dgbmv* and *zgbmv*. It also exists with a native C interface as *cblas_sgbmv*.

3.3.39 sgemv

sgemv performs one of the matrix-vector operations

```
y := alpha*A*x + beta*y,   or   y := alpha*A**T*x + beta*y,
```

where alpha and beta are scalars, x and y are vectors and A is an m by n matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgemv(TRANS, M, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
```

C specification:

```
#include "armpl.h"

void sgemv_(const char *trans, const armpl_int_t *m, const armpl_int_t *n,
            const float *alpha, const float *a, const armpl_int_t *lda,
            const float *x, const armpl_int_t *incx, const float *beta,
            float *y, const armpl_int_t *incy, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' y := alpha*A*x + beta*y.

TRANS = 'T' or 't' y := alpha*A^T *x + beta*y.

TRANS = 'C' or 'c' y := alpha*A^T *x + beta*y.

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of the matrix A. M must be at least zero.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix A. N must be at least zero.

ALPHA Input parameter.

ALPHA is REAL

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). Before entry, the leading m by n part of the array A must contain the matrix of coefficients.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, m).

X Input parameter.

X is REAL

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)) when TRANS = 'N' or 'n' and at least (1 + (m - 1)*abs(INCX)) otherwise. Before entry, the incremented array X must contain the vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

BETA Input parameter.

BETA is REAL

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input.

Y Input and output parameter.

Y is REAL

Y is an array, dimension at least. (1 + (m - 1)*abs(INCY)) when TRANS = 'N' or 'n' and at least (1 + (n - 1)*abs(INCY)) otherwise. Before entry with BETA non-zero, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

Related Information

For this routine in other precisions, please see *cgemv*, *dgemv* and *zgemv*. It also exists with a native C interface as *cblas_sgemv*.

3.3.40 sger

sger performs the rank 1 operation

```
A := alpha*x*y**T + A,
```

where alpha is a scalar, x is an m element vector, y is an n element vector and A is an m by n matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine sger(M, N, ALPHA, X, INCX, Y, INCY, A, LDA)
```

C specification:

```
#include "armpl.h"

void sger_(const armpl_int_t *m, const armpl_int_t *n, const float *alpha,
           const float *x, const armpl_int_t *incx, const float *y,
           const armpl_int_t *incy, float *a, const armpl_int_t *lda);
```

Parameters

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of the matrix A. M must be at least zero.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix A. N must be at least zero.

ALPHA Input parameter.

ALPHA is REAL

On entry, ALPHA specifies the scalar alpha.

X Input parameter.

X is REAL

X is an array, dimension at least. (1 + (m - 1)*abs(INCX)). Before entry, the incremented array X must contain the m element vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Y Input parameter.

Y is REAL

Y is an array, dimension at least. (1 + (n - 1)*abs(INCY)). Before entry, the incremented array Y must contain the n element vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). Before entry, the leading m by n part of the array A must contain the matrix of coefficients. On exit, A is overwritten by the updated matrix.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, m).

Related Information

For this routine in other precisions, please see *dger*. It also exists with a native C interface as *cblas_sger*.

3.3.41 ssbmv

ssbmv performs the matrix-vector operation

```
y := alpha*A*x + beta*y,
```

where alpha and beta are scalars, x and y are n element vectors and A is an n by n symmetric band matrix, with k super-diagonals.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssbmv(UPLO, N, K, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
```

C specification:

```
#include "armpl.h"

void ssbmv_(const char *uplo, const armpl_int_t *n, const armpl_int_t *k,
            const float *alpha, const float *a, const armpl_int_t *lda,
            const float *x, const armpl_int_t *incx, const float *beta,
            float *y, const armpl_int_t *incy, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the band matrix A is being supplied as follows:

UPLO = 'U' or 'u' The upper triangular part of A is being supplied.

UPLO = 'L' or 'l' The lower triangular part of A is being supplied.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

K Input parameter.

K is INTEGER

On entry, K specifies the number of super-diagonals of the matrix A. K must satisfy 0 .le. K.

ALPHA Input parameter.

ALPHA is REAL

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). Before entry with UPLO = 'U' or 'u', the leading (k + 1) by n part of the array A must contain the upper triangular band part of the symmetric matrix, supplied column by column, with the leading diagonal of the matrix in row (k + 1) of the array, the first super-diagonal starting at position 2 in row k, and so on. The top left k by k triangle of the array A is not referenced. The following program segment will transfer the upper triangular part of a symmetric band matrix from conventional full matrix storage to band storage:

```
   DO 20, J = 1, N
      K = ``KU`` + 1 - J
      DO 10, I = MAX( 1, J - ``K`` ), J
         A( M + I, J ) = matrix( I, J )
10    CONTINUE
20 CONTINUE
```

Before entry with UPLO = 'L' or 'l', the leading (k + 1) by n part of the array A must contain the lower triangular band part of the symmetric matrix, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k by k triangle of the array A is not referenced. The following program segment will transfer the lower triangular part of a symmetric band matrix from conventional full matrix storage to band storage:

```
   DO 20, J = 1, N
      M = 1 - J
      DO 10, I = J, MIN( ``N``, J + ``K`` )
         A( M + I, J ) = matrix( I, J )
10    CONTINUE
20 CONTINUE
```

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least (k + 1).

X Input parameter.

X is REAL

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

BETA Input parameter.

BETA is REAL

On entry, BETA specifies the scalar beta.

Y Input and output parameter.

Y is REAL

Y is an array, dimension at least. (1 + (n - 1)*abs(INCY)). Before entry, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

Related Information

For this routine in other precisions, please see *dsbmv*. It also exists with a native C interface as *cblas_ssbmv*.

3.3.42 sspmv

sspmv performs the matrix-vector operation

```
y := alpha*A*x + beta*y,
```

where alpha and beta are scalars, x and y are n element vectors and A is an n by n symmetric matrix, supplied in packed form.

Syntax

Fortran specification:

```
use armpl_library

subroutine sspmv(UPLO, N, ALPHA, AP, X, INCX, BETA, Y, INCY)
```

C specification:

```
#include "armpl.h"

void sspmv_(const char *uplo, const armpl_int_t *n, const float *alpha,
            const float *ap, const float *x, const armpl_int_t *incx,
            const float *beta, float *y, const armpl_int_t *incy, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array AP as follows:

UPLO = 'U' or 'u' The upper triangular part of A is supplied in AP.

UPLO = 'L' or 'l' The lower triangular part of A is supplied in AP.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

ALPHA Input parameter.

ALPHA is REAL

On entry, ALPHA specifies the scalar alpha.

AP Input parameter.

AP is REAL

AP is an array, dimension at least. ((n*(n + 1))/2). Before entry with UPLO = 'U' or 'u', the array AP must contain the upper triangular part of the symmetric matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(1, 2) and a(2, 2) respectively, and so on. Before entry with UPLO = 'L' or 'l', the array AP must contain the lower triangular part of the symmetric matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(2, 1) and a(3, 1) respectively, and so on.

X Input parameter.

X is REAL

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

BETA Input parameter.

BETA is REAL

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input.

Y Input and output parameter.

Y is REAL

Y is an array, dimension at least. (1 + (n - 1)*abs(INCY)). Before entry, the incremented array Y must contain the n element vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

Related Information

For this routine in other precisions, please see *cspmv*, *dspmv* and *zspmv*. It also exists with a native C interface as *cblas_sspmv*.

3.3.43 sspr

sspr performs the symmetric rank 1 operation

```
A := alpha*x*x**T + A,
```

where alpha is a real scalar, x is an n element vector and A is an n by n symmetric matrix, supplied in packed form.

Syntax

Fortran specification:

```
use armpl_library

subroutine sspr(UPLO, N, ALPHA, X, INCX, AP)
```

C specification:

```
#include "armpl.h"

void sspr_(const char *uplo, const armpl_int_t *n, const float *alpha,
           const float *x, const armpl_int_t *incx, float *ap, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array AP as follows:

UPLO = 'U' or 'u' The upper triangular part of A is supplied in AP.

UPLO = 'L' or 'l' The lower triangular part of A is supplied in AP.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

ALPHA Input parameter.

ALPHA is REAL

On entry, ALPHA specifies the scalar alpha.

X Input parameter.

X is REAL

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

AP Input and output parameter.

AP is REAL

AP is an array, dimension at least. ((n*(n + 1))/2). Before entry with UPLO = 'U' or 'u', the array AP must contain the upper triangular part of the symmetric matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(1, 2) and a(2, 2) respectively, and so on. On exit, the array AP is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the array AP must contain the lower triangular part of the symmetric matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(2, 1) and a(3, 1) respectively, and so on. On exit, the array AP is overwritten by the lower triangular part of the updated matrix.

Related Information

For this routine in other precisions, please see *cspr*, *dspr* and *zspr*. It also exists with a native C interface as *cblas_sspr*.

3.3.44 sspr2

sspr2 performs the symmetric rank 2 operation

```
A := alpha*x*y**T + alpha*y*x**T + A,
```

where alpha is a scalar, x and y are n element vectors and A is an n by n symmetric matrix, supplied in packed form.

Syntax

Fortran specification:

```
use armpl_library

subroutine sspr2(UPLO, N, ALPHA, X, INCX, Y, INCY, AP)
```

C specification:

```
#include "armpl.h"

void sspr2_(const char *uplo, const armpl_int_t *n, const float *alpha,
            const float *x, const armpl_int_t *incx, const float *y,
            const armpl_int_t *incy, float *ap, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array AP as follows:

UPLO = 'U' or 'u' The upper triangular part of A is supplied in AP.

UPLO = 'L' or 'l' The lower triangular part of A is supplied in AP.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

ALPHA Input parameter.

ALPHA is REAL

On entry, ALPHA specifies the scalar alpha.

X Input parameter.

X is REAL

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Y Input parameter.

Y is REAL

Y is an array, dimension at least. (1 + (n - 1)*abs(INCY)). Before entry, the incremented array Y must contain the n element vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

AP Input and output parameter.

AP is REAL

AP is an array, dimension at least. ((n*(n + 1))/2). Before entry with UPLO = 'U' or 'u', the array AP must contain the upper triangular part of the symmetric matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(1, 2) and a(2, 2) respectively, and so on. On exit, the array AP is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the array AP must contain the lower triangular part of the symmetric matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(2, 1) and a(3, 1) respectively, and so on. On exit, the array AP is overwritten by the lower triangular part of the updated matrix.

Related Information

For this routine in other precisions, please see *dspr2*. It also exists with a native C interface as *cblas_sspr2*.

3.3.45 ssymv

ssymv performs the matrix-vector operation

```
y := alpha*A*x + beta*y,
```

where alpha and beta are scalars, x and y are n element vectors and A is an n by n symmetric matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssymv(UPLO, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
```

C specification:

```
#include "armpl.h"

void ssymv_(const char *uplo, const armpl_int_t *n, const float *alpha,
            const float *a, const armpl_int_t *lda, const float *x,
            const armpl_int_t *incx, const float *beta, float *y,
            const armpl_int_t *incy, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the array A is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of A is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of A is to be referenced.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

ALPHA Input parameter.

ALPHA is REAL

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not referenced. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, n).

X Input parameter.

X is REAL

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

BETA Input parameter.

BETA is REAL

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input.

Y Input and output parameter.

Y is REAL

Y is an array, dimension at least. (1 + (n - 1)*abs(INCY)). Before entry, the incremented array Y must contain the n element vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

Related Information

For this routine in other precisions, please see *csymv*, *dsymv* and *zsymv*. It also exists with a native C interface as *cblas_ssymv*.

3.3.46 ssyr

ssyr performs the symmetric rank 1 operation

```
A := alpha*x*x**T + A,
```

where alpha is a real scalar, x is an n element vector and A is an n by n symmetric matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssyr(UPLO, N, ALPHA, X, INCX, A, LDA)
```

C specification:

```
#include "armpl.h"

void ssyr_(const char *uplo, const armpl_int_t *n, const float *alpha,
           const float *x, const armpl_int_t *incx, float *a,
           const armpl_int_t *lda, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the array A is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of A is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of A is to be referenced.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

ALPHA Input parameter.

ALPHA is REAL

On entry, ALPHA specifies the scalar alpha.

X Input parameter.

X is REAL

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not referenced. On exit, the upper triangular part of the array A is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced. On exit, the lower triangular part of the array A is overwritten by the lower triangular part of the updated matrix.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, n).

Related Information

For this routine in other precisions, please see *csyr*, *dsyr* and *zsyr*. It also exists with a native C interface as *cblas_ssyr*.

3.3.47 ssyr2

ssyr2 performs the symmetric rank 2 operation

```
A := alpha*x*y**T + alpha*y*x**T + A,
```

where alpha is a scalar, x and y are n element vectors and A is an n by n symmetric matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssyr2(UPLO, N, ALPHA, X, INCX, Y, INCY, A, LDA)
```

C specification:

```
#include "armpl.h"

void ssyr2_(const char *uplo, const armpl_int_t *n, const float *alpha,
            const float *x, const armpl_int_t *incx, const float *y,
            const armpl_int_t *incy, float *a, const armpl_int_t *lda, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the array A is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of A is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of A is to be referenced.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

ALPHA Input parameter.

ALPHA is REAL

On entry, ALPHA specifies the scalar alpha.

X Input parameter.

X is REAL

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Y Input parameter.

Y is REAL

Y is an array, dimension at least. (1 + (n - 1)*abs(INCY)). Before entry, the incremented array Y must contain the n element vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not referenced. On exit, the upper triangular part of the array A is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced. On exit, the lower triangular part of the array A is overwritten by the lower triangular part of the updated matrix.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, n).

Related Information

For this routine in other precisions, please see *dsyr2*. It also exists with a native C interface as *cblas_ssyr2*.

3.3.48 stbmv

stbmv performs one of the matrix-vector operations

```
x := A*x,   or   x := A**T*x,
```

where x is an n element vector and A is an n by n unit, or non-unit, upper or lower triangular band matrix, with (k + 1) diagonals.

Syntax

Fortran specification:

```
use armpl_library

subroutine stbmv(UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX)
```

C specification:

```
#include "armpl.h"

void stbmv_(const char *uplo, const char *trans, const char *diag,
            const armpl_int_t *n, const armpl_int_t *k, const float *a,
            const armpl_int_t *lda, float *x, const armpl_int_t *incx, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' x := A*x.

TRANS = 'T' or 't' x := A^T *x.

TRANS = 'C' or 'c' x := A^T *x.

DIAG Input parameter.

DIAG is CHARACTER*1

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

K Input parameter.

K is INTEGER

On entry with UPLO = 'U' or 'u', K specifies the number of super-diagonals of the matrix A. On entry with UPLO = 'L' or 'l', K specifies the number of sub-diagonals of the matrix A. K must satisfy 0 .le. K.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). Before entry with UPLO = 'U' or 'u', the leading (k + 1) by n part of the array A must contain the upper triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row (k + 1) of the array, the first super-diagonal starting at position 2 in row k, and so on. The top left k by k triangle of the array A is not referenced. The following program segment will transfer an upper triangular band matrix from conventional full matrix storage to band storage:

```
   DO 20, J = 1, N
      M = ``K`` + 1 - J
      DO 10, I = MAX( 1, J - ``K`` ), J
         A( M + I, J ) = matrix( I, J )
10    CONTINUE
20 CONTINUE
```

Before entry with UPLO = 'L' or 'l', the leading (k + 1) by n part of the array A must contain the lower triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k by k triangle of the array A is not referenced. The following program segment will transfer a lower triangular band matrix from conventional full matrix storage to band storage:

```
   DO 20, J = 1, N
      M = 1 - J
      DO 10, I = J, MIN( ``N``, J + ``K`` )
         A( M + I, J ) = matrix( I, J )
10    CONTINUE
20 CONTINUE
```

Note that when DIAG = 'U' or 'u' the elements of the array A corresponding to the diagonal elements of the matrix are not referenced, but are assumed to be unity.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least (k + 1).

X Input and output parameter.

X is REAL

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x. On exit, X is overwritten with the transformed vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Related Information

For this routine in other precisions, please see *ctbmv*, *dtbmv* and *ztbmv*. It also exists with a native C interface as *cblas_stbmv*.

3.3.49 stbsv

stbsv solves one of the systems of equations

```
A*x = b,   or   A**T*x = b,
```

where b and x are n element vectors and A is an n by n unit, or non-unit, upper or lower triangular band matrix, with (k + 1) diagonals.

No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

Syntax

Fortran specification:

```
use armpl_library

subroutine stbsv(UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX)
```

C specification:

```
#include "armpl.h"

void stbsv_(const char *uplo, const char *trans, const char *diag,
            const armpl_int_t *n, const armpl_int_t *k, const float *a,
            const armpl_int_t *lda, float *x, const armpl_int_t *incx, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the equations to be solved as follows:

TRANS = 'N' or 'n' A*x = b.

TRANS = 'T' or 't' A^T *x = b.

TRANS = 'C' or 'c' A^T *x = b.

DIAG Input parameter.

DIAG is CHARACTER*1

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

K Input parameter.

K is INTEGER

On entry with UPLO = 'U' or 'u', K specifies the number of super-diagonals of the matrix A. On entry with UPLO = 'L' or 'l', K specifies the number of sub-diagonals of the matrix A. K must satisfy 0 .le. K.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). Before entry with UPLO = 'U' or 'u', the leading (k + 1) by n part of the array A must contain the upper triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row (k + 1) of the array, the first super-diagonal starting at position 2 in row k, and so on. The top left k by k triangle of the array A is not referenced. The following program segment will transfer an upper triangular band matrix from conventional full matrix storage to band storage:

```
   DO 20, J = 1, N
      M = ``K`` + 1 - J
      DO 10, I = MAX( 1, J - ``K`` ), J
         A( M + I, J ) = matrix( I, J )
10    CONTINUE
20 CONTINUE
```

Before entry with UPLO = 'L' or 'l', the leading (k + 1) by n part of the array A must contain the lower triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k by k triangle of the array A is not referenced. The following program segment will transfer a lower triangular band matrix from conventional full matrix storage to band storage:

```
   DO 20, J = 1, N
      M = 1 - J
      DO 10, I = J, MIN( ``N``, J + ``K`` )
         A( M + I, J ) = matrix( I, J )
10    CONTINUE
20 CONTINUE
```

Note that when DIAG = 'U' or 'u' the elements of the array A corresponding to the diagonal elements of the matrix are not referenced, but are assumed to be unity.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least (k + 1).

X Input and output parameter.

X is REAL

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element right-hand side vector b. On exit, X is overwritten with the solution vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Related Information

For this routine in other precisions, please see *ctbsv*, *dtbsv* and *ztbsv*. It also exists with a native C interface as *cblas_stbsv*.

3.3.50 stpmv

stpmv performs one of the matrix-vector operations

```
x := A*x,   or   x := A**T*x,
```

where x is an n element vector and A is an n by n unit, or non-unit, upper or lower triangular matrix, supplied in packed form.

Syntax

Fortran specification:

```
use armpl_library

subroutine stpmv(UPLO, TRANS, DIAG, N, AP, X, INCX)
```

C specification:

```
#include "armpl.h"

void stpmv_(const char *uplo, const char *trans, const char *diag,
            const armpl_int_t *n, const float *ap, float *x,
            const armpl_int_t *incx, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' x := A*x.

TRANS = 'T' or 't' x := A^T *x.

TRANS = 'C' or 'c' x := A^T *x.

DIAG Input parameter.

DIAG is CHARACTER*1

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

AP Input parameter.

AP is REAL

AP is an array, dimension at least. ((n*(n + 1))/2). Before entry with UPLO = 'U' or 'u', the array AP must contain the upper triangular matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(1, 2) and a(2, 2) respectively, and so on. Before entry with UPLO = 'L' or 'l', the array AP must contain the lower triangular matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(2, 1) and a(3, 1) respectively, and so on. Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced, but are assumed to be unity.

X Input and output parameter.

X is REAL

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x. On exit, X is overwritten with the transformed vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Related Information

For this routine in other precisions, please see *ctpmv*, *dtpmv* and *ztpmv*. It also exists with a native C interface as *cblas_stpmv*.

3.3.51 stpsv

stpsv solves one of the systems of equations

```
A*x = b,   or   A**T*x = b,
```

where b and x are n element vectors and A is an n by n unit, or non-unit, upper or lower triangular matrix, supplied in packed form.

No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

Syntax

Fortran specification:

```
use armpl_library

subroutine stpsv(UPLO, TRANS, DIAG, N, AP, X, INCX)
```

C specification:

```
#include "armpl.h"

void stpsv_(const char *uplo, const char *trans, const char *diag,
            const armpl_int_t *n, const float *ap, float *x,
            const armpl_int_t *incx, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the equations to be solved as follows:

TRANS = 'N' or 'n' A*x = b.

TRANS = 'T' or 't' A^T *x = b.

TRANS = 'C' or 'c' A^T *x = b.

DIAG Input parameter.

DIAG is CHARACTER*1

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

AP Input parameter.

AP is REAL

AP is an array, dimension at least. ((n*(n + 1))/2). Before entry with UPLO = 'U' or 'u', the array AP must contain the upper triangular matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(1, 2) and a(2, 2) respectively, and so on. Before entry with UPLO = 'L' or 'l', the array AP must contain the lower triangular matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(2, 1) and a(3, 1) respectively, and so on. Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced, but are assumed to be unity.

X Input and output parameter.

X is REAL

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element right-hand side vector b. On exit, X is overwritten with the solution vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Related Information

For this routine in other precisions, please see *ctpsv*, *dtpsv* and *ztpsv*. It also exists with a native C interface as *cblas_stpsv*.

3.3.52 strmv

strmv performs one of the matrix-vector operations

```
x := A*x,   or   x := A**T*x,
```

where x is an n element vector and A is an n by n unit, or non-unit, upper or lower triangular matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine strmv(UPLO, TRANS, DIAG, N, A, LDA, X, INCX)
```

C specification:

```
#include "armpl.h"

void strmv_(const char *uplo, const char *trans, const char *diag,
            const armpl_int_t *n, const float *a, const armpl_int_t *lda,
            float *x, const armpl_int_t *incx, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' x := A*x.

TRANS = 'T' or 't' x := A^T *x.

TRANS = 'C' or 'c' x := A^T *x.

DIAG Input parameter.

DIAG is CHARACTER*1

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular matrix and the strictly lower triangular part of A is not referenced. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular matrix and the strictly upper triangular part of A is not referenced. Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, n).

X Input and output parameter.

X is REAL

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x. On exit, X is overwritten with the transformed vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Related Information

For this routine in other precisions, please see *ctrmv*, *dtrmv* and *ztrmv*. It also exists with a native C interface as *cblas_strmv*.

3.3.53 strsv

strsv solves one of the systems of equations

```
A*x = b,   or   A**T*x = b,
```

where b and x are n element vectors and A is an n by n unit, or non-unit, upper or lower triangular matrix.

No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

Syntax

Fortran specification:

```
use armpl_library

subroutine strsv(UPLO, TRANS, DIAG, N, A, LDA, X, INCX)
```

C specification:

```
#include "armpl.h"

void strsv_(const char *uplo, const char *trans, const char *diag,
            const armpl_int_t *n, const float *a, const armpl_int_t *lda,
            float *x, const armpl_int_t *incx, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the equations to be solved as follows:

TRANS = 'N' or 'n' A*x = b.

TRANS = 'T' or 't' A^T *x = b.

TRANS = 'C' or 'c' A^T *x = b.

DIAG Input parameter.

DIAG is CHARACTER*1

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular matrix and the strictly lower triangular part of A is not referenced. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular matrix and the strictly upper triangular part of A is not referenced. Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, n).

X Input and output parameter.

X is REAL

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element right-hand side vector b. On exit, X is overwritten with the solution vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Related Information

For this routine in other precisions, please see *ctrsv*, *dtrsv* and *ztrsv*. It also exists with a native C interface as *cblas_strsv*.

3.3.54 zgbmv

zgbmv performs one of the matrix-vector operations

```
 y := alpha*A*x + beta*y,   or   y := alpha*A**T*x + beta*y,   or

y := alpha*A**H*x + beta*y,
```

where alpha and beta are scalars, x and y are vectors and A is an m by n band matrix, with kl sub-diagonals and ku super-diagonals.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgbmv(TRANS, M, N, KL, KU, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
```

C specification:

```
#include "armpl.h"

void zgbmv_(const char *trans, const armpl_int_t *m, const armpl_int_t *n,
            const armpl_int_t *kl, const armpl_int_t *ku,
            const armpl_doublecomplex_t *alpha,
            const armpl_doublecomplex_t *a, const armpl_int_t *lda,
            const armpl_doublecomplex_t *x, const armpl_int_t *incx,
            const armpl_doublecomplex_t *beta, armpl_doublecomplex_t *y,
            const armpl_int_t *incy, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' y := alpha*A*x + beta*y.

TRANS = 'T' or 't' y := alpha*A^T *x + beta*y.

TRANS = 'C' or 'c' y := alpha*A^H *x + beta*y.

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of the matrix A. M must be at least zero.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix A. N must be at least zero.

KL Input parameter.

KL is INTEGER

On entry, KL specifies the number of sub-diagonals of the matrix A. KL must satisfy 0 .le. KL.

KU Input parameter.

KU is INTEGER

On entry, KU specifies the number of super-diagonals of the matrix A. KU must satisfy 0 .le. KU.

ALPHA Input parameter.

ALPHA is COMPLEX*16

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). Before entry, the leading (kl + ku + 1) by n part of the array A must contain the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row (ku + 1) of the array, the first super-diagonal starting at position 2 in row ku, the first sub-diagonal starting at position 1 in row (ku + 2), and so on. Elements in the array A that do not correspond to elements in the band matrix (such as the top left ku by ku triangle) are not referenced. The following program segment will transfer a band matrix from conventional full matrix storage to band storage:

```
   DO 20, J = 1, N
      K = ``KU`` + 1 - J
      DO 10, I = MAX( 1, J - ``KU`` ), MIN( ``M``, J + ``KL`` )
         A( K + I, J ) = matrix( I, J )
10    CONTINUE
20 CONTINUE
```

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least (kl + ku + 1).

X Input parameter.

X is COMPLEX*16

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)) when TRANS = 'N' or 'n' and at least (1 + (m - 1)*abs(INCX)) otherwise. Before entry, the incremented array X must contain the vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

BETA Input parameter.

BETA is COMPLEX*16

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input.

Y Input and output parameter.

Y is COMPLEX*16

Y is an array, dimension at least. (1 + (m - 1)*abs(INCY)) when TRANS = 'N' or 'n' and at least (1 + (n - 1)*abs(INCY)) otherwise. Before entry, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

Related Information

For this routine in other precisions, please see *cgbmv*, *dgbmv* and *sgbmv*. It also exists with a native C interface as *cblas_zgbmv*.

3.3.55 zgemv

zgemv performs one of the matrix-vector operations

```
 y := alpha*A*x + beta*y,   or   y := alpha*A**T*x + beta*y,   or

y := alpha*A**H*x + beta*y,
```

where alpha and beta are scalars, x and y are vectors and A is an m by n matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgemv(TRANS, M, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
```

C specification:

```
#include "armpl.h"

void zgemv_(const char *trans, const armpl_int_t *m, const armpl_int_t *n,
            const armpl_doublecomplex_t *alpha,
            const armpl_doublecomplex_t *a, const armpl_int_t *lda,
            const armpl_doublecomplex_t *x, const armpl_int_t *incx,
            const armpl_doublecomplex_t *beta, armpl_doublecomplex_t *y,
            const armpl_int_t *incy, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' y := alpha*A*x + beta*y.

TRANS = 'T' or 't' y := alpha*A^T *x + beta*y.

TRANS = 'C' or 'c' y := alpha*A^H *x + beta*y.

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of the matrix A. M must be at least zero.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix A. N must be at least zero.

ALPHA Input parameter.

ALPHA is COMPLEX*16

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). Before entry, the leading m by n part of the array A must contain the matrix of coefficients.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, m).

X Input parameter.

X is COMPLEX*16

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)) when TRANS = 'N' or 'n' and at least (1 + (m - 1)*abs(INCX)) otherwise. Before entry, the incremented array X must contain the vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

BETA Input parameter.

BETA is COMPLEX*16

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input.

Y Input and output parameter.

Y is COMPLEX*16

Y is an array, dimension at least. (1 + (m - 1)*abs(INCY)) when TRANS = 'N' or 'n' and at least (1 + (n - 1)*abs(INCY)) otherwise. Before entry with BETA non-zero, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

Related Information

For this routine in other precisions, please see *cgemv*, *dgemv* and *sgemv*. It also exists with a native C interface as *cblas_zgemv*.

3.3.56 zgerc

zgerc performs the rank 1 operation

```
A := alpha*x*y**H + A,
```

where alpha is a scalar, x is an m element vector, y is an n element vector and A is an m by n matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgerc(M, N, ALPHA, X, INCX, Y, INCY, A, LDA)
```

C specification:

```
#include "armpl.h"

void zgerc_(const armpl_int_t *m, const armpl_int_t *n,
            const armpl_doublecomplex_t *alpha,
            const armpl_doublecomplex_t *x, const armpl_int_t *incx,
            const armpl_doublecomplex_t *y, const armpl_int_t *incy,
            armpl_doublecomplex_t *a, const armpl_int_t *lda);
```

Parameters

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of the matrix A. M must be at least zero.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix A. N must be at least zero.

ALPHA Input parameter.

ALPHA is COMPLEX*16

On entry, ALPHA specifies the scalar alpha.

X Input parameter.

X is COMPLEX*16

X is an array, dimension at least. (1 + (m - 1)*abs(INCX)). Before entry, the incremented array X must contain the m element vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Y Input parameter.

Y is COMPLEX*16

Y is an array, dimension at least. (1 + (n - 1)*abs(INCY)). Before entry, the incremented array Y must contain the n element vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). Before entry, the leading m by n part of the array A must contain the matrix of coefficients. On exit, A is overwritten by the updated matrix.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, m).

Related Information

For this routine in other precisions, please see *cgerc*. It also exists with a native C interface as *cblas_zgerc*.

3.3.57 zgeru

zgeru performs the rank 1 operation

```
A := alpha*x*y**T + A,
```

where alpha is a scalar, x is an m element vector, y is an n element vector and A is an m by n matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgeru(M, N, ALPHA, X, INCX, Y, INCY, A, LDA)
```

C specification:

```
#include "armpl.h"

void zgeru_(const armpl_int_t *m, const armpl_int_t *n,
            const armpl_doublecomplex_t *alpha,
            const armpl_doublecomplex_t *x, const armpl_int_t *incx,
            const armpl_doublecomplex_t *y, const armpl_int_t *incy,
            armpl_doublecomplex_t *a, const armpl_int_t *lda);
```

Parameters

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of the matrix A. M must be at least zero.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix A. N must be at least zero.

ALPHA Input parameter.

ALPHA is COMPLEX*16

On entry, ALPHA specifies the scalar alpha.

X Input parameter.

X is COMPLEX*16

X is an array, dimension at least. (1 + (m - 1)*abs(INCX)). Before entry, the incremented array X must contain the m element vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Y Input parameter.

Y is COMPLEX*16

Y is an array, dimension at least. (1 + (n - 1)*abs(INCY)). Before entry, the incremented array Y must contain the n element vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). Before entry, the leading m by n part of the array A must contain the matrix of coefficients. On exit, A is overwritten by the updated matrix.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, m).

Related Information

For this routine in other precisions, please see *cgeru*. It also exists with a native C interface as *cblas_zgeru*.

3.3.58 zhbmv

zhbmv performs the matrix-vector operation

```
y := alpha*A*x + beta*y,
```

where alpha and beta are scalars, x and y are n element vectors and A is an n by n hermitian band matrix, with k super-diagonals.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhbmv(UPLO, N, K, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
```

C specification:

```
#include "armpl.h"

void zhbmv_(const char *uplo, const armpl_int_t *n, const armpl_int_t *k,
            const armpl_doublecomplex_t *alpha,
            const armpl_doublecomplex_t *a, const armpl_int_t *lda,
            const armpl_doublecomplex_t *x, const armpl_int_t *incx,
            const armpl_doublecomplex_t *beta, armpl_doublecomplex_t *y,
            const armpl_int_t *incy, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the band matrix A is being supplied as follows:

UPLO = 'U' or 'u' The upper triangular part of A is being supplied.

UPLO = 'L' or 'l' The lower triangular part of A is being supplied.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

K Input parameter.

K is INTEGER

On entry, K specifies the number of super-diagonals of the matrix A. K must satisfy 0 .le. K.

ALPHA Input parameter.

ALPHA is COMPLEX*16

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). Before entry with UPLO = 'U' or 'u', the leading (k + 1) by n part of the array A must contain the upper triangular band part of the hermitian matrix, supplied column by column, with the leading diagonal of the matrix in row (k + 1) of the array, the first super-diagonal starting at position 2 in row k, and so on. The top left k by k triangle of the array A is not referenced. The following program segment will transfer the upper triangular part of a hermitian band matrix from conventional full matrix storage to band storage:

```
   DO 20, J = 1, N
      M = ``K`` + 1 - J
      DO 10, I = MAX( 1, J - ``K`` ), J
         A( M + I, J ) = matrix( I, J )
10    CONTINUE
20 CONTINUE
```

Before entry with UPLO = 'L' or 'l', the leading (k + 1) by n part of the array A must contain the lower triangular band part of the hermitian matrix, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k by k triangle of the array A is not referenced. The following program segment will transfer the lower triangular part of a hermitian band matrix from conventional full matrix storage to band storage:

```
   DO 20, J = 1, N
      M = 1 - J
      DO 10, I = J, MIN( ``N``, J + ``K`` )
         A( M + I, J ) = matrix( I, J )
10    CONTINUE
20 CONTINUE
```

Note that the imaginary parts of the diagonal elements need not be set and are assumed to be zero.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least (k + 1).

X Input parameter.

X is COMPLEX*16

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

BETA Input parameter.

BETA is COMPLEX*16

On entry, BETA specifies the scalar beta.

Y Input and output parameter.

Y is COMPLEX*16

Y is an array, dimension at least. (1 + (n - 1)*abs(INCY)). Before entry, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

Related Information

For this routine in other precisions, please see *chbmv*. It also exists with a native C interface as *cblas_zhbmv*.

3.3.59 zhemv

zhemv performs the matrix-vector operation

```
y := alpha*A*x + beta*y,
```

where alpha and beta are scalars, x and y are n element vectors and A is an n by n hermitian matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhemv(UPLO, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
```

C specification:

```
#include "armpl.h"

void zhemv_(const char *uplo, const armpl_int_t *n,
            const armpl_doublecomplex_t *alpha,
            const armpl_doublecomplex_t *a, const armpl_int_t *lda,
            const armpl_doublecomplex_t *x, const armpl_int_t *incx,
            const armpl_doublecomplex_t *beta, armpl_doublecomplex_t *y,
            const armpl_int_t *incy, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the array A is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of A is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of A is to be referenced.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

ALPHA Input parameter.

ALPHA is COMPLEX*16

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the hermitian matrix and the strictly lower triangular part of A is not referenced. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the hermitian matrix and the strictly upper triangular part of A is not referenced. Note that the imaginary parts of the diagonal elements need not be set and are assumed to be zero.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, n).

X Input parameter.

X is COMPLEX*16

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

BETA Input parameter.

BETA is COMPLEX*16

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input.

Y Input and output parameter.

Y is COMPLEX*16

Y is an array, dimension at least. (1 + (n - 1)*abs(INCY)). Before entry, the incremented array Y must contain the n element vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

Related Information

For this routine in other precisions, please see *chemv*. It also exists with a native C interface as *cblas_zhemv*.

3.3.60 zher

zher performs the hermitian rank 1 operation

```
A := alpha*x*x**H + A,
```

where alpha is a real scalar, x is an n element vector and A is an n by n hermitian matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine zher(UPLO, N, ALPHA, X, INCX, A, LDA)
```

C specification:

```
#include "armpl.h"

void zher_(const char *uplo, const armpl_int_t *n, const double *alpha,
           const armpl_doublecomplex_t *x, const armpl_int_t *incx,
           armpl_doublecomplex_t *a, const armpl_int_t *lda, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the array A is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of A is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of A is to be referenced.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

ALPHA Input parameter.

ALPHA is DOUBLE PRECISION.

On entry, ALPHA specifies the scalar alpha.

X Input parameter.

X is COMPLEX*16

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the hermitian matrix and the strictly lower triangular part of A is not referenced. On exit, the upper triangular part of the array A is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the hermitian matrix and the strictly upper triangular part of A is not referenced. On exit, the lower triangular part of the array A is overwritten by the lower triangular part of the updated matrix. Note that the imaginary parts of the diagonal elements need not be set, they are assumed to be zero, and on exit they are set to zero.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, n).

Related Information

For this routine in other precisions, please see *cher*. It also exists with a native C interface as *cblas_zher*.

3.3.61 zher2

zher2 performs the hermitian rank 2 operation

```
A := alpha*x*y**H + conjg( alpha )*y*x**H + A,
```

where alpha is a scalar, x and y are n element vectors and A is an n by n hermitian matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine zher2(UPLO, N, ALPHA, X, INCX, Y, INCY, A, LDA)
```

C specification:

```
#include "armpl.h"

void zher2_(const char *uplo, const armpl_int_t *n,
            const armpl_doublecomplex_t *alpha,
            const armpl_doublecomplex_t *x, const armpl_int_t *incx,
            const armpl_doublecomplex_t *y, const armpl_int_t *incy,
            armpl_doublecomplex_t *a, const armpl_int_t *lda, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the array A is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of A is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of A is to be referenced.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

ALPHA Input parameter.

ALPHA is COMPLEX*16

On entry, ALPHA specifies the scalar alpha.

X Input parameter.

X is COMPLEX*16

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Y Input parameter.

Y is COMPLEX*16

Y is an array, dimension at least. (1 + (n - 1)*abs(INCY)). Before entry, the incremented array Y must contain the n element vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the hermitian matrix and the strictly lower triangular part of A is not referenced. On exit, the upper triangular part of the array A is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the hermitian matrix and the strictly upper triangular part of A is not referenced. On exit, the lower triangular part of the array A is overwritten by the lower triangular part of the updated matrix. Note that the imaginary parts of the diagonal elements need not be set, they are assumed to be zero, and on exit they are set to zero.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, n).

Related Information

For this routine in other precisions, please see *cher2*. It also exists with a native C interface as *cblas_zher2*.

3.3.62 zhpmv

zhpmv performs the matrix-vector operation

```
y := alpha*A*x + beta*y,
```

where alpha and beta are scalars, x and y are n element vectors and A is an n by n hermitian matrix, supplied in packed form.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhpmv(UPLO, N, ALPHA, AP, X, INCX, BETA, Y, INCY)
```

C specification:

```
#include "armpl.h"

void zhpmv_(const char *uplo, const armpl_int_t *n,
            const armpl_doublecomplex_t *alpha,
            const armpl_doublecomplex_t *a, const armpl_doublecomplex_t *x,
            const armpl_int_t *incx, const armpl_doublecomplex_t *beta,
            armpl_doublecomplex_t *y, const armpl_int_t *incy, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array AP as follows:

UPLO = 'U' or 'u' The upper triangular part of A is supplied in AP.

UPLO = 'L' or 'l' The lower triangular part of A is supplied in AP.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

ALPHA Input parameter.

ALPHA is COMPLEX*16

On entry, ALPHA specifies the scalar alpha.

AP Input parameter.

AP is COMPLEX*16

AP is an array, dimension at least. ((n*(n + 1))/2). Before entry with UPLO = 'U' or 'u', the array AP must contain the upper triangular part of the hermitian matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(1, 2) and a(2, 2) respectively, and so on. Before entry with UPLO = 'L' or 'l', the array AP must contain the lower triangular part of the hermitian matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(2, 1) and a(3, 1) respectively, and so on. Note that the imaginary parts of the diagonal elements need not be set and are assumed to be zero.

X Input parameter.

X is COMPLEX*16

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

BETA Input parameter.

BETA is COMPLEX*16

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input.

Y Input and output parameter.

Y is COMPLEX*16

Y is an array, dimension at least. (1 + (n - 1)*abs(INCY)). Before entry, the incremented array Y must contain the n element vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

Related Information

For this routine in other precisions, please see *chpmv*. It also exists with a native C interface as *cblas_zhpmv*.

3.3.63 zhpr

zhpr performs the hermitian rank 1 operation

```
A := alpha*x*x**H + A,
```

where alpha is a real scalar, x is an n element vector and A is an n by n hermitian matrix, supplied in packed form.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhpr(UPLO, N, ALPHA, X, INCX, AP)
```

C specification:

```
#include "armpl.h"

void zhpr_(const char *uplo, const armpl_int_t *n, const double *alpha,
           const armpl_doublecomplex_t *x, const armpl_int_t *incx,
           armpl_doublecomplex_t *a, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array AP as follows:

UPLO = 'U' or 'u' The upper triangular part of A is supplied in AP.

UPLO = 'L' or 'l' The lower triangular part of A is supplied in AP.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

ALPHA Input parameter.

ALPHA is DOUBLE PRECISION.

On entry, ALPHA specifies the scalar alpha.

X Input parameter.

X is COMPLEX*16

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

AP Input and output parameter.

AP is COMPLEX*16

AP is an array, dimension at least. ((n*(n + 1))/2). Before entry with UPLO = 'U' or 'u', the array AP must contain the upper triangular part of the hermitian matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(1, 2) and a(2, 2) respectively, and so on. On exit, the array AP is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the array AP must contain the lower triangular part of the hermitian matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(2, 1) and a(3, 1) respectively, and so on. On exit, the array AP is overwritten by the lower triangular part of the updated matrix. Note that the imaginary parts of the diagonal elements need not be set, they are assumed to be zero, and on exit they are set to zero.

Related Information

For this routine in other precisions, please see *chpr*. It also exists with a native C interface as *cblas_zhpr*.

3.3.64 zhpr2

zhpr2 performs the hermitian rank 2 operation

```
A := alpha*x*y**H + conjg( alpha )*y*x**H + A,
```

where alpha is a scalar, x and y are n element vectors and A is an n by n hermitian matrix, supplied in packed form.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhpr2(UPLO, N, ALPHA, X, INCX, Y, INCY, AP)
```

C specification:

```
#include "armpl.h"

void zhpr2_(const char *uplo, const armpl_int_t *n,
            const armpl_doublecomplex_t *alpha,
            const armpl_doublecomplex_t *x, const armpl_int_t *incx,
            const armpl_doublecomplex_t *y, const armpl_int_t *incy,
            armpl_doublecomplex_t *a, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array AP as follows:

UPLO = 'U' or 'u' The upper triangular part of A is supplied in AP.

UPLO = 'L' or 'l' The lower triangular part of A is supplied in AP.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

ALPHA Input parameter.

ALPHA is COMPLEX*16

On entry, ALPHA specifies the scalar alpha.

X Input parameter.

X is COMPLEX*16

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Y Input parameter.

Y is COMPLEX*16

Y is an array, dimension at least. (1 + (n - 1)*abs(INCY)). Before entry, the incremented array Y must contain the n element vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

AP Input and output parameter.

AP is COMPLEX*16

AP is an array, dimension at least. ((n*(n + 1))/2). Before entry with UPLO = 'U' or 'u', the array AP must contain the upper triangular part of the hermitian matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(1, 2) and a(2, 2) respectively, and so on. On exit, the array AP is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the array AP must contain the lower triangular part of the hermitian matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(2, 1) and a(3, 1) respectively, and so on. On exit, the array AP is overwritten by the lower triangular part of the updated matrix. Note that the imaginary parts of the diagonal elements need not be set, they are assumed to be zero, and on exit they are set to zero.

Related Information

For this routine in other precisions, please see *chpr2*. It also exists with a native C interface as *cblas_zhpr2*.

3.3.65 zspmv

zspmv performs the matrix-vector operation

```
y := alpha*A*x + beta*y,
```

where alpha and beta are scalars, x and y are n element vectors and A is an n by n symmetric matrix, supplied in packed form.

Syntax

Fortran specification:

```
use armpl_library

subroutine zspmv(UPLO, N, ALPHA, AP, X, INCX, BETA, Y, INCY)
```

C specification:

```
#include "armpl.h"

void zspmv_(const char *uplo, const armpl_int_t *n,
            const armpl_doublecomplex_t *alpha,
            const armpl_doublecomplex_t *ap, const armpl_doublecomplex_t *x,
            const armpl_int_t *incx, const armpl_doublecomplex_t *beta,
            armpl_doublecomplex_t *y, const armpl_int_t *incy, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array AP as follows:

UPLO = 'U' or 'u' The upper triangular part of A is supplied in AP.

UPLO = 'L' or 'l' The lower triangular part of A is supplied in AP.

Unchanged on exit.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero. Unchanged on exit.

ALPHA Input parameter.

ALPHA is COMPLEX*16

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

AP Input parameter.

AP is COMPLEX*16

AP is an array, dimension at least. ((N*(N + 1))/2). Before entry, with UPLO = 'U' or 'u', the array AP must contain the upper triangular part of the symmetric matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(1, 2) and a(2, 2) respectively, and so on. Before entry, with UPLO = 'L' or 'l', the array AP must contain the lower triangular part of the symmetric matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(2, 1) and a(3, 1) respectively, and so on. Unchanged on exit.

X Input parameter.

X is COMPLEX*16

X is an array, dimension at least. (1 + (N - 1)*abs(INCX)). Before entry, the incremented array X must contain the N- element vector x. Unchanged on exit.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

BETA Input parameter.

BETA is COMPLEX*16

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input. Unchanged on exit.

Y Input and output parameter.

Y is COMPLEX*16

Y is an array, dimension at least. (1 + (N - 1)*abs(INCY)). Before entry, the incremented array Y must contain the n element vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero. Unchanged on exit.

Related Information

For this routine in other precisions, please see *cspmv*, *dspmv* and *sspmv*.

3.3.66 zspr

zspr performs the symmetric rank 1 operation

```
A := alpha*x*x**H + A,
```

where alpha is a complex scalar, x is an n element vector and A is an n by n symmetric matrix, supplied in packed form.

Syntax

Fortran specification:

```
use armpl_library

subroutine zspr(UPLO, N, ALPHA, X, INCX, AP)
```

C specification:

```
#include "armpl.h"

void zspr_(const char *uplo, const armpl_int_t *n,
           const armpl_doublecomplex_t *alpha, const armpl_doublecomplex_t *x,
           const armpl_int_t *incx, armpl_doublecomplex_t *ap, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array AP as follows:

UPLO = 'U' or 'u' The upper triangular part of A is supplied in AP.

UPLO = 'L' or 'l' The lower triangular part of A is supplied in AP.

Unchanged on exit.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero. Unchanged on exit.

ALPHA Input parameter.

ALPHA is COMPLEX*16

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

X Input parameter.

X is COMPLEX*16

X is an array, dimension at least. (1 + (N - 1)*abs(INCX)). Before entry, the incremented array X must contain the N- element vector x. Unchanged on exit.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

AP Input and output parameter.

AP is COMPLEX*16

AP is an array, dimension at least. ((N*(N + 1))/2). Before entry, with UPLO = 'U' or 'u', the array AP must contain the upper triangular part of the symmetric matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(1, 2) and a(2, 2) respectively, and so on. On exit, the array AP is overwritten by the upper triangular part of the updated matrix. Before entry, with UPLO = 'L' or 'l', the array AP must contain the lower triangular part of the symmetric matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(2, 1) and a(3, 1) respectively, and so on. On exit, the array AP is overwritten by the lower triangular part of the updated matrix. Note that the imaginary parts of the diagonal elements need not be set, they are assumed to be zero, and on exit they are set to zero.

Related Information

For this routine in other precisions, please see *cspr*, *dspr* and *sspr*.

3.3.67 zsymv

zsymv performs the matrix-vector operation

```
y := alpha*A*x + beta*y,
```

where alpha and beta are scalars, x and y are n element vectors and A is an n by n symmetric matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsymv(UPLO, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
```

C specification:

```
#include "armpl.h"

void zsymv_(const char *uplo, const armpl_int_t *n,
            const armpl_doublecomplex_t *alpha,
            const armpl_doublecomplex_t *a, const armpl_int_t *lda,
            const armpl_doublecomplex_t *x, const armpl_int_t *incx,
            const armpl_doublecomplex_t *beta, armpl_doublecomplex_t *y,
            const armpl_int_t *incy, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the array A is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of A is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of A is to be referenced.

Unchanged on exit.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero. Unchanged on exit.

ALPHA Input parameter.

ALPHA is COMPLEX*16

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). Before entry, with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not referenced. Before entry, with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced. Unchanged on exit.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, N). Unchanged on exit.

X Input parameter.

X is COMPLEX*16

X is an array, dimension at least. (1 + (N - 1)*abs(INCX)). Before entry, the incremented array X must contain the N- element vector x. Unchanged on exit.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

BETA Input parameter.

BETA is COMPLEX*16

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input. Unchanged on exit.

Y Input and output parameter.

Y is COMPLEX*16

Y is an array, dimension at least. (1 + (N - 1)*abs(INCY)). Before entry, the incremented array Y must contain the n element vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero. Unchanged on exit.

Related Information

For this routine in other precisions, please see *csymv*, *dsymv* and *ssymv*.

3.3.68 zsyr

zsyr performs the symmetric rank 1 operation

```
A := alpha*x*x**H + A,
```

where alpha is a complex scalar, x is an n element vector and A is an n by n symmetric matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsyr(UPLO, N, ALPHA, X, INCX, A, LDA)
```

C specification:

```
#include "armpl.h"

void zsyr_(const char *uplo, const armpl_int_t *n,
           const armpl_doublecomplex_t *alpha, const armpl_doublecomplex_t *x,
           const armpl_int_t *incx, armpl_doublecomplex_t *a,
           const armpl_int_t *lda, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the array A is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of A is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of A is to be referenced.

Unchanged on exit.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero. Unchanged on exit.

ALPHA Input parameter.

ALPHA is COMPLEX*16

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

X Input parameter.

X is COMPLEX*16

X is an array, dimension at least. (1 + (N - 1)*abs(INCX)). Before entry, the incremented array X must contain the N- element vector x. Unchanged on exit.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). Before entry, with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not referenced. On exit, the upper triangular part of the array A is overwritten by the upper triangular part of the updated matrix. Before entry, with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced. On exit, the lower triangular part of the array A is overwritten by the lower triangular part of the updated matrix.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, N). Unchanged on exit.

Related Information

For this routine in other precisions, please see *csyr*, *dsyr* and *ssyr*. It also exists with a native C interface as *LAPACKE_zsyr*.

3.3.69 ztbmv

ztbmv performs one of the matrix-vector operations

```
x := A*x,   or   x := A**T*x,   or   x := A**H*x,
```

where x is an n element vector and A is an n by n unit, or non-unit, upper or lower triangular band matrix, with (k + 1) diagonals.

Syntax

Fortran specification:

```
use armpl_library

subroutine ztbmv(UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX)
```

C specification:

```
#include "armpl.h"

void ztbmv_(const char *uplo, const char *trans, const char *diag,
            const armpl_int_t *n, const armpl_int_t *k,
            const armpl_doublecomplex_t *a, const armpl_int_t *lda,
            armpl_doublecomplex_t *x, const armpl_int_t *incx, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' x := A*x.

TRANS = 'T' or 't' x := A^T *x.

TRANS = 'C' or 'c' x := A^H *x.

DIAG Input parameter.

DIAG is CHARACTER*1

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

K Input parameter.

K is INTEGER

On entry with UPLO = 'U' or 'u', K specifies the number of super-diagonals of the matrix A. On entry with UPLO = 'L' or 'l', K specifies the number of sub-diagonals of the matrix A. K must satisfy 0 .le. K.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N).. Before entry with UPLO = 'U' or 'u', the leading (k + 1) by n part of the array A must contain the upper triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row (k + 1) of the array, the first super-diagonal starting at position 2 in row k, and so on. The top left k by k triangle of the array A is not referenced. The following program segment will transfer an upper triangular band matrix from conventional full matrix storage to band storage:

```
   DO 20, J = 1, N
      M = ``K`` + 1 - J
      DO 10, I = MAX( 1, J - ``K`` ), J
         A( M + I, J ) = matrix( I, J )
10    CONTINUE
20 CONTINUE
```

Before entry with UPLO = 'L' or 'l', the leading (k + 1) by n part of the array A must contain the lower triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k by k triangle of the array A is not referenced. The following program segment will transfer a lower triangular band matrix from conventional full matrix storage to band storage:

```
   DO 20, J = 1, N
      M = 1 - J
      DO 10, I = J, MIN( ``N``, J + ``K`` )
         A( M + I, J ) = matrix( I, J )
10    CONTINUE
20 CONTINUE
```

Note that when DIAG = 'U' or 'u' the elements of the array A corresponding to the diagonal elements of the matrix are not referenced, but are assumed to be unity.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least (k + 1).

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x. On exit, X is overwritten with the transformed vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Related Information

For this routine in other precisions, please see *ctbmv*, *dtbmv* and *stbmv*. It also exists with a native C interface as *cblas_ztbmv*.

3.3.70 ztbsv

ztbsv solves one of the systems of equations

```
A*x = b,   or   A**T*x = b,   or   A**H*x = b,
```

where b and x are n element vectors and A is an n by n unit, or non-unit, upper or lower triangular band matrix, with (k + 1) diagonals.

No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

Syntax

Fortran specification:

```
use armpl_library

subroutine ztbsv(UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX)
```

C specification:

```
#include "armpl.h"

void ztbsv_(const char *uplo, const char *trans, const char *diag,
            const armpl_int_t *n, const armpl_int_t *k,
            const armpl_doublecomplex_t *a, const armpl_int_t *lda,
            armpl_doublecomplex_t *x, const armpl_int_t *incx, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the equations to be solved as follows:

TRANS = 'N' or 'n' A*x = b.

TRANS = 'T' or 't' A^T *x = b.

TRANS = 'C' or 'c' A^H *x = b.

DIAG Input parameter.

DIAG is CHARACTER*1

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

K Input parameter.

K is INTEGER

On entry with UPLO = 'U' or 'u', K specifies the number of super-diagonals of the matrix A. On entry with UPLO = 'L' or 'l', K specifies the number of sub-diagonals of the matrix A. K must satisfy 0 .le. K.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). Before entry with UPLO = 'U' or 'u', the leading (k + 1) by n part of the array A must contain the upper triangular band part of the matrix of coefficients, supplied column by column,

with the leading diagonal of the matrix in row (k + 1) of the array, the first super-diagonal starting at position 2 in row k, and so on. The top left k by k triangle of the array A is not referenced. The following program segment will transfer an upper triangular band matrix from conventional full matrix storage to band storage:

```
   DO 20, J = 1, N
      M = ``K`` + 1 - J
      DO 10, I = MAX( 1, J - ``K`` ), J
         A( M + I, J ) = matrix( I, J )
10    CONTINUE
20 CONTINUE
```

Before entry with UPLO = 'L' or 'l', the leading (k + 1) by n part of the array A must contain the lower triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k by k triangle of the array A is not referenced. The following program segment will transfer a lower triangular band matrix from conventional full matrix storage to band storage:

```
   DO 20, J = 1, N
      M = 1 - J
      DO 10, I = J, MIN( ``N``, J + ``K`` )
         A( M + I, J ) = matrix( I, J )
10    CONTINUE
20 CONTINUE
```

Note that when DIAG = 'U' or 'u' the elements of the array A corresponding to the diagonal elements of the matrix are not referenced, but are assumed to be unity.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least (k + 1).

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element right-hand side vector b. On exit, X is overwritten with the solution vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Related Information

For this routine in other precisions, please see *ctbsv*, *dtbsv* and *stbsv*. It also exists with a native C interface as *cblas_ztbsv*.

3.3.71 ztpmv

ztpmv performs one of the matrix-vector operations

```
x := A*x,   or   x := A**T*x,   or   x := A**H*x,
```

where x is an n element vector and A is an n by n unit, or non-unit, upper or lower triangular matrix, supplied in packed form.

Syntax

Fortran specification:

```
use armpl_library

subroutine ztpmv(UPLO, TRANS, DIAG, N, AP, X, INCX)
```

C specification:

```
#include "armpl.h"

void ztpmv_(const char *uplo, const char *trans, const char *diag,
            const armpl_int_t *n, const armpl_doublecomplex_t *ap,
            armpl_doublecomplex_t *x, const armpl_int_t *incx, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' x := A*x.

TRANS = 'T' or 't' x := A^T *x.

TRANS = 'C' or 'c' x := A^H *x.

DIAG Input parameter.

DIAG is CHARACTER*1

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

AP Input parameter.

AP is COMPLEX*16

AP is an array, dimension at least. ((n*(n + 1))/2). Before entry with UPLO = 'U' or 'u', the array AP must contain the upper triangular matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(1, 2) and a(2, 2) respectively, and so on. Before entry with UPLO = 'L' or 'l', the array AP must contain the lower triangular matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(2, 1) and a(3, 1) respectively, and so on. Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced, but are assumed to be unity.

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x. On exit, X is overwritten with the transformed vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Related Information

For this routine in other precisions, please see *ctpmv*, *dtpmv* and *stpmv*. It also exists with a native C interface as *cblas_ztpmv*.

3.3.72 ztpsv

ztpsv solves one of the systems of equations

```
A*x = b,   or   A**T*x = b,   or   A**H*x = b,
```

where b and x are n element vectors and A is an n by n unit, or non-unit, upper or lower triangular matrix, supplied in packed form.

No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

Syntax

Fortran specification:

```
use armpl_library

subroutine ztpsv(UPLO, TRANS, DIAG, N, AP, X, INCX)
```

C specification:

```
#include "armpl.h"

void ztpsv_(const char *uplo, const char *trans, const char *diag,
            const armpl_int_t *n, const armpl_doublecomplex_t *ap,
            armpl_doublecomplex_t *x, const armpl_int_t *incx, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the equations to be solved as follows:

TRANS = 'N' or 'n' A*x = b.

TRANS = 'T' or 't' A^T *x = b.

TRANS = 'C' or 'c' A^H *x = b.

DIAG Input parameter.

DIAG is CHARACTER*1

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

AP Input parameter.

AP is COMPLEX*16

AP is an array, dimension at least. ((n*(n + 1))/2). Before entry with UPLO = 'U' or 'u', the array AP must contain the upper triangular matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(1, 2) and a(2, 2) respectively, and so on. Before entry with UPLO = 'L' or 'l', the array AP must contain the lower triangular matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(2, 1) and a(3, 1) respectively, and so on. Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced, but are assumed to be unity.

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element right-hand side vector b. On exit, X is overwritten with the solution vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Related Information

For this routine in other precisions, please see *ctpsv*, *dtpsv* and *stpsv*. It also exists with a native C interface as *cblas_ztpsv*.

3.3.73 ztrmv

ztrmv performs one of the matrix-vector operations

```
x := A*x,   or   x := A**T*x,   or   x := A**H*x,
```

where x is an n element vector and A is an n by n unit, or non-unit, upper or lower triangular matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine ztrmv(UPLO, TRANS, DIAG, N, A, LDA, X, INCX)
```

C specification:

```
#include "armpl.h"

void ztrmv_(const char *uplo, const char *trans, const char *diag,
            const armpl_int_t *n, const armpl_doublecomplex_t *a,
            const armpl_int_t *lda, armpl_doublecomplex_t *x,
            const armpl_int_t *incx, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' x := A*x.

TRANS = 'T' or 't' x := A^T *x.

TRANS = 'C' or 'c' x := A^H *x.

DIAG Input parameter.

DIAG is CHARACTER*1

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N).. Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular matrix and the strictly lower triangular part of A is not referenced. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular matrix and the strictly upper triangular part of A is not referenced. Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, n).

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x. On exit, X is overwritten with the transformed vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Related Information

For this routine in other precisions, please see *ctrmv*, *dtrmv* and *strmv*. It also exists with a native C interface as *cblas_ztrmv*.

3.3.74 ztrsv

ztrsv solves one of the systems of equations

```
A*x = b,   or   A**T*x = b,   or   A**H*x = b,
```

where b and x are n element vectors and A is an n by n unit, or non-unit, upper or lower triangular matrix.

No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

Syntax

Fortran specification:

```
use armpl_library

subroutine ztrsv(UPLO, TRANS, DIAG, N, A, LDA, X, INCX)
```

C specification:

```
#include "armpl.h"

void ztrsv_(const char *uplo, const char *trans, const char *diag,
            const armpl_int_t *n, const armpl_doublecomplex_t *a,
            const armpl_int_t *lda, armpl_doublecomplex_t *x,
            const armpl_int_t *incx, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the equations to be solved as follows:

TRANS = 'N' or 'n' A*x = b.

TRANS = 'T' or 't' A^T *x = b.

TRANS = 'C' or 'c' A^H *x = b.

DIAG Input parameter.

DIAG is CHARACTER*1

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix A. N must be at least zero.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular matrix and the strictly lower triangular part of A is not referenced. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular matrix and the strictly upper triangular part of A is not referenced. Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, n).

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element right-hand side vector b. On exit, X is overwritten with the solution vector x.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

Related Information

For this routine in other precisions, please see *ctrsv*, *dtrsv* and *strsv*. It also exists with a native C interface as *cblas_ztrsv*.

Level 2 BLAS routines are typically memory-vector operations and are normally memory bound. All functions are detailed in armpl.h.

3.4 BLAS level 3

3.4.1 cgemm

cgemm performs one of the matrix-matrix operations

```
C := alpha*op( A )*op( B ) + beta*C,
```

where op(X) is one of

```
op( X ) = X   or   op( X ) = X**T   or   op( X ) = X**H,
```

alpha and beta are scalars, and A, B and C are matrices, with op(A) an m by k matrix, op(B) a k by n matrix and C an m by n matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgemm(TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C,
                 LDC)
```

C specification:

```
#include "armpl.h"

void cgemm_(const char *transa, const char *transb, const armpl_int_t *m,
            const armpl_int_t *n, const armpl_int_t *k,
            const armpl_singlecomplex_t *alpha,
            const armpl_singlecomplex_t *a, const armpl_int_t *lda,
            const armpl_singlecomplex_t *b, const armpl_int_t *ldb,
            const armpl_singlecomplex_t *beta, armpl_singlecomplex_t *c,
            const armpl_int_t *ldc, ... );
```

Parameters

TRANSA Input parameter.

TRANSA is CHARACTER*1

On entry, TRANSA specifies the form of op(A) to be used in the matrix multiplication as follows:

TRANSA = 'N' or 'n', op(A) = A.

TRANSA = 'T' or 't', op(A) = A^T .

TRANSA = 'C' or 'c', op(A) = A^H .

TRANSB Input parameter.

TRANSB is CHARACTER*1

On entry, TRANSB specifies the form of op(B) to be used in the matrix multiplication as follows:

TRANSB = 'N' or 'n', op(B) = B.

TRANSB = 'T' or 't', op(B) = B^T .

TRANSB = 'C' or 'c', op(B) = B^H .

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of the matrix op(A) and of the matrix C. M must be at least zero.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix op(B) and the number of columns of the matrix C. N must be at least zero.

K Input parameter.

K is INTEGER

On entry, K specifies the number of columns of the matrix op(A) and the number of rows of the matrix op(B). K must be at least zero.

ALPHA Input parameter.

ALPHA is COMPLEX

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, ka), where ka is. k when TRANSA = 'N' or 'n', and is m otherwise. Before entry with TRANSA = 'N' or 'n', the leading m by k part of the array A must contain the matrix A, otherwise the leading k by m part of the array A must contain the matrix A.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANSA = 'N' or 'n' then LDA must be at least max(1, m), otherwise LDA must be at least max(1, k).

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, kb), where kb is. n when TRANSB = 'N' or 'n', and is k otherwise. Before entry with TRANSB = 'N' or 'n', the leading k by n part of the array B must contain the matrix B, otherwise the leading n by k part of the array B must contain the matrix B.

LDB Input parameter.

LDB is INTEGER

On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. When TRANSB = 'N' or 'n' then LDB must be at least max(1, k), otherwise LDB must be at least max(1, n).

BETA Input parameter.

BETA is COMPLEX

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then C need not be set on input.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). Before entry, the leading m by n part of the array C must contain the matrix C, except when beta is zero, in which case C need not be set on entry. On exit, the array C is overwritten by the m by n matrix (alpha*op(A)*op(B) + beta* C).

LDC Input parameter.

LDC is INTEGER

On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max(1, m).

Related Information

For this routine in other precisions, please see *dgemm*, *hgemm*, *sgemm* and *zgemm*. It also exists with a native C interface as *cblas_cgemm*.

3.4.2 chemm

chemm performs one of the matrix-matrix operations

```
C := alpha*A*B + beta*C,
```

or

```
C := alpha*B*A + beta*C,
```

where alpha and beta are scalars, A is an hermitian matrix and B and C are m by n matrices.

Syntax

Fortran specification:

```
use armpl_library

subroutine chemm(SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
```

C specification:

```
#include "armpl.h"

void chemm_(const char *side, const char *uplo, const armpl_int_t *m,
            const armpl_int_t *n, const armpl_singlecomplex_t *alpha,
            const armpl_singlecomplex_t *a, const armpl_int_t *lda,
            const armpl_singlecomplex_t *b, const armpl_int_t *ldb,
            const armpl_singlecomplex_t *beta, armpl_singlecomplex_t *c,
            const armpl_int_t *ldc, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

On entry, SIDE specifies whether the hermitian matrix A appears on the left or right in the operation as follows:

SIDE = 'L' or 'l' C := alpha*A* B + beta*C,

SIDE = 'R' or 'r' C := alpha*B* A + beta*C,

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the hermitian matrix A is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of the hermitian matrix is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of the hermitian matrix is to be referenced.

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of the matrix C. M must be at least zero.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix C. N must be at least zero.

ALPHA Input parameter.

ALPHA is COMPLEX

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, ka), where ka is. m when SIDE = 'L' or 'l' and is n otherwise. Before entry with SIDE = 'L' or 'l', the m by m part of the array A must contain the hermitian matrix, such that when UPLO = 'U' or 'u', the leading m by m upper triangular part of the array A must contain the upper triangular part of the hermitian matrix and the strictly lower triangular part of A is not referenced, and when UPLO = 'L' or 'l', the leading m by m lower triangular part of the array A must contain the lower triangular part of the hermitian matrix and the strictly upper triangular part of A is not referenced. Before entry with SIDE = 'R' or 'r', the n by n part of the array A must contain the hermitian matrix, such that when UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the hermitian matrix and the strictly lower triangular part of A is not referenced, and when UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the hermitian matrix and the strictly upper triangular part of A is not referenced. Note that the imaginary parts of the diagonal elements need not be set, they are assumed to be zero.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When SIDE = 'L' or 'l' then LDA must be at least max(1, m), otherwise LDA must be at least max(1, n).

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, N). Before entry, the leading m by n part of the array B must contain the matrix B.

LDB Input parameter.

LDB is INTEGER

On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. LDB must be at least max(1, m).

BETA Input parameter.

BETA is COMPLEX

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then C need not be set on input.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). Before entry, the leading m by n part of the array C must contain the matrix C, except when beta is zero, in which case C need not be set on entry. On exit, the array C is overwritten by the m by n updated matrix.

LDC Input parameter.

LDC is INTEGER

On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max(1, m).

Related Information

For this routine in other precisions, please see *zhemm*. It also exists with a native C interface as *cblas_chemm*.

3.4.3 cher2k

cher2k performs one of the hermitian rank 2k operations

```
C := alpha*A*B**H + conjg( alpha )*B*A**H + beta*C,
```

or

```
C := alpha*A**H*B + conjg( alpha )*B**H*A + beta*C,
```

where alpha and beta are scalars with beta real, C is an n by n hermitian matrix and A and B are n by k matrices in the first case and k by n matrices in the second case.

Syntax

Fortran specification:

```
use armpl_library

subroutine cher2k(UPLO, TRANS, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
```

C specification:

```
#include "armpl.h"

void cher2k_(const char *uplo, const char *trans, const armpl_int_t *n,
             const armpl_int_t *k, const armpl_singlecomplex_t *alpha,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const armpl_singlecomplex_t *b, const armpl_int_t *ldb,
             const float *beta, armpl_singlecomplex_t *c,
             const armpl_int_t *ldc, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the array C is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of C is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of C is to be referenced.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' C := alpha*A*B^H + conjg(alpha)*B*A^H + beta*C.

TRANS = 'C' or 'c' C := alpha*A^H * B + conjg(alpha)*B^H * A + beta*C.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix C. N must be at least zero.

K Input parameter.

K is INTEGER

On entry with TRANS = 'N' or 'n', K specifies the number of columns of the matrices A and B, and on entry with TRANS = 'C' or 'c', K specifies the number of rows of the matrices A and B. K must be at least zero.

ALPHA Input parameter.

ALPHA is COMPLEX

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, ka), where ka is. k when TRANS = 'N' or 'n', and is n otherwise. Before entry with TRANS = 'N' or 'n', the leading n by k part of the array A must contain the matrix A, otherwise the leading k by n part of the array A must contain the matrix A.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDA must be at least max(1, n), otherwise LDA must be at least max(1, k).

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, kb), where kb is. k when TRANS = 'N' or 'n', and is n otherwise. Before entry with TRANS = 'N' or 'n', the leading n by k part of the array B must contain the matrix B, otherwise the leading k by n part of the array B must contain the matrix B.

LDB Input parameter.

LDB is INTEGER

On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDB must be at least max(1, n), otherwise LDB must be at least max(1, k).

BETA Input parameter.

BETA is REAL

On entry, BETA specifies the scalar beta.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array C must contain the upper triangular part of the hermitian matrix and the strictly lower triangular part of C is not referenced. On exit, the upper triangular part of the array C is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array C must contain the lower triangular part of the hermitian matrix and the strictly upper triangular part of C is not referenced. On exit, the lower triangular part of the array C is overwritten by the lower triangular part of the updated matrix. Note that the imaginary parts of the diagonal elements need not be set, they are assumed to be zero, and on exit they are set to zero.

LDC Input parameter.

LDC is INTEGER

On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max(1, n).

Related Information

For this routine in other precisions, please see *zher2k*. It also exists with a native C interface as *cblas_cher2k*.

3.4.4 cherk

`cherk` performs one of the hermitian rank k operations

```
C := alpha*A*A**H + beta*C,
```

or

```
C := alpha*A**H*A + beta*C,
```

where alpha and beta are real scalars, C is an n by n hermitian matrix and A is an n by k matrix in the first case and a k by n matrix in the second case.

Syntax

Fortran specification:

```
use armpl_library

subroutine cherk(UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C, LDC)
```

C specification:

```
#include "armpl.h"

void cherk_(const char *uplo, const char *trans, const armpl_int_t *n,
            const armpl_int_t *k, const float *alpha,
            const armpl_singlecomplex_t *a, const armpl_int_t *lda,
            const float *beta, armpl_singlecomplex_t *c,
            const armpl_int_t *ldc, ... );
```

Parameters

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

On entry, `UPLO` specifies whether the upper or lower triangular part of the array `C` is to be referenced as follows:

`UPLO` = 'U' or 'u' Only the upper triangular part of C is to be referenced.

`UPLO` = 'L' or 'l' Only the lower triangular part of C is to be referenced.

TRANS Input parameter.

`TRANS` is `CHARACTER*1`

On entry, `TRANS` specifies the operation to be performed as follows:

`TRANS` = 'N' or 'n' `C` := alpha*A*A^H + beta*C.

`TRANS` = 'C' or 'c' `C` := alpha*A^H * `A` + beta*C.

N Input parameter.

`N` is `INTEGER`

On entry, `N` specifies the order of the matrix `C`. `N` must be at least zero.

K Input parameter.

`K` is `INTEGER`

On entry with `TRANS` = 'N' or 'n', `K` specifies the number of columns of the matrix `A`, and on entry with `TRANS` = 'C' or 'c', `K` specifies the number of rows of the matrix `A`. `K` must be at least zero.

ALPHA Input parameter.

ALPHA is REAL

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, ka), where ka is. k when TRANS = 'N' or 'n', and is n otherwise. Before entry with TRANS = 'N' or 'n', the leading n by k part of the array A must contain the matrix A, otherwise the leading k by n part of the array A must contain the matrix A.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDA must be at least max(1, n), otherwise LDA must be at least max(1, k).

BETA Input parameter.

BETA is REAL

On entry, BETA specifies the scalar beta.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array C must contain the upper triangular part of the hermitian matrix and the strictly lower triangular part of C is not referenced. On exit, the upper triangular part of the array C is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array C must contain the lower triangular part of the hermitian matrix and the strictly upper triangular part of C is not referenced. On exit, the lower triangular part of the array C is overwritten by the lower triangular part of the updated matrix. Note that the imaginary parts of the diagonal elements need not be set, they are assumed to be zero, and on exit they are set to zero.

LDC Input parameter.

LDC is INTEGER

On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max(1, n).

Related Information

For this routine in other precisions, please see *zherk*. It also exists with a native C interface as *cblas_cherk*.

3.4.5 csymm

csymm performs one of the matrix-matrix operations

```
C := alpha*A*B + beta*C,
```

or

```
C := alpha*B*A + beta*C,
```

where alpha and beta are scalars, A is a symmetric matrix and B and C are m by n matrices.

Syntax

Fortran specification:

```
use armpl_library

subroutine csymm(SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
```

C specification:

```
#include "armpl.h"

void csymm_(const char *side, const char *uplo, const armpl_int_t *m,
            const armpl_int_t *n, const armpl_singlecomplex_t *alpha,
            const armpl_singlecomplex_t *a, const armpl_int_t *lda,
            const armpl_singlecomplex_t *b, const armpl_int_t *ldb,
            const armpl_singlecomplex_t *beta, armpl_singlecomplex_t *c,
            const armpl_int_t *ldc, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

On entry, SIDE specifies whether the symmetric matrix A appears on the left or right in the operation as follows:

SIDE = 'L' or 'l' C := alpha*A* B + beta*C,

SIDE = 'R' or 'r' C := alpha*B* A + beta*C,

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the symmetric matrix A is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of the symmetric matrix is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of the symmetric matrix is to be referenced.

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of the matrix C. M must be at least zero.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix C. N must be at least zero.

ALPHA Input parameter.

ALPHA is COMPLEX

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, ka), where ka is. m when SIDE = 'L' or 'l' and is n otherwise. Before entry with SIDE = 'L' or 'l', the m by m part of the array A must contain the symmetric matrix, such that when UPLO = 'U' or 'u', the leading m by m upper triangular part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not referenced, and when UPLO =

'L' or 'l', the leading m by m lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced. Before entry with SIDE = 'R' or 'r', the n by n part of the array A must contain the symmetric matrix, such that when UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not referenced, and when UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When SIDE = 'L' or 'l' then LDA must be at least max(1, m), otherwise LDA must be at least max(1, n).

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, N). Before entry, the leading m by n part of the array B must contain the matrix B.

LDB Input parameter.

LDB is INTEGER

On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. LDB must be at least max(1, m).

BETA Input parameter.

BETA is COMPLEX

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then C need not be set on input.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). Before entry, the leading m by n part of the array C must contain the matrix C, except when beta is zero, in which case C need not be set on entry. On exit, the array C is overwritten by the m by n updated matrix.

LDC Input parameter.

LDC is INTEGER

On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max(1, m).

Related Information

For this routine in other precisions, please see *dsymm*, *ssymm* and *zsymm*. It also exists with a native C interface as *cblas_csymm*.

3.4.6 csyr2k

csyr2k performs one of the symmetric rank 2k operations

```
C := alpha*A*B**T + alpha*B*A**T + beta*C,
```

or

```
C := alpha*A**T*B + alpha*B**T*A + beta*C,
```

where alpha and beta are scalars, C is an n by n symmetric matrix and A and B are n by k matrices in the first case and k by n matrices in the second case.

Syntax

Fortran specification:

```
use armpl_library

subroutine csyr2k(UPLO, TRANS, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
```

C specification:

```
#include "armpl.h"

void csyr2k_(const char *uplo, const char *trans, const armpl_int_t *n,
             const armpl_int_t *k, const armpl_singlecomplex_t *alpha,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const armpl_singlecomplex_t *b, const armpl_int_t *ldb,
             const armpl_singlecomplex_t *beta, armpl_singlecomplex_t *c,
             const armpl_int_t *ldc, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the array C is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of C is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of C is to be referenced.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' C := alpha*A*B^T + alpha*B*A^T + beta*C.

TRANS = 'T' or 't' C := alpha*A^T * B + alpha*B^T * A + beta*C.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix C. N must be at least zero.

K Input parameter.

K is INTEGER

On entry with TRANS = 'N' or 'n', K specifies the number of columns of the matrices A and B, and on entry with TRANS = 'T' or 't', K specifies the number of rows of the matrices A and B. K must be at least zero.

ALPHA Input parameter.

ALPHA is COMPLEX

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, ka), where ka is. k when TRANS = 'N' or 'n', and is n otherwise. Before entry with TRANS = 'N' or 'n', the leading n by k part of the array A must contain the matrix A, otherwise the leading k by n part of the array A must contain the matrix A.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDA must be at least max(1, n), otherwise LDA must be at least max(1, k).

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, kb), where kb is. k when TRANS = 'N' or 'n', and is n otherwise. Before entry with TRANS = 'N' or 'n', the leading n by k part of the array B must contain the matrix B, otherwise the leading k by n part of the array B must contain the matrix B.

LDB Input parameter.

LDB is INTEGER

On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDB must be at least max(1, n), otherwise LDB must be at least max(1, k).

BETA Input parameter.

BETA is COMPLEX

On entry, BETA specifies the scalar beta.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array C must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of C is not referenced. On exit, the upper triangular part of the array C is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array C must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of C is not referenced. On exit, the lower triangular part of the array C is overwritten by the lower triangular part of the updated matrix.

LDC Input parameter.

LDC is INTEGER

On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max(1, n).

Related Information

For this routine in other precisions, please see *dsyr2k*, *ssyr2k* and *zsyr2k*. It also exists with a native C interface as *cblas_csyr2k*.

3.4.7 csyrk

csyrk performs one of the symmetric rank k operations

```
C := alpha*A*A**T + beta*C,
```

or

```
C := alpha*A**T*A + beta*C,
```

where alpha and beta are scalars, C is an n by n symmetric matrix and A is an n by k matrix in the first case and a k by n matrix in the second case.

Syntax

Fortran specification:

```
use armpl_library

subroutine csyrk(UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C, LDC)
```

C specification:

```
#include "armpl.h"

void csyrk_(const char *uplo, const char *trans, const armpl_int_t *n,
            const armpl_int_t *k, const armpl_singlecomplex_t *alpha,
            const armpl_singlecomplex_t *a, const armpl_int_t *lda,
            const armpl_singlecomplex_t *beta, armpl_singlecomplex_t *c,
            const armpl_int_t *ldc, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the array C is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of C is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of C is to be referenced.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' C := alpha*A*A^T + beta*C.

TRANS = 'T' or 't' C := alpha*A^T * A + beta*C.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix C. N must be at least zero.

K Input parameter.

K is INTEGER

On entry with TRANS = 'N' or 'n', K specifies the number of columns of the matrix A, and on entry with TRANS = 'T' or 't', K specifies the number of rows of the matrix A. K must be at least zero.

ALPHA Input parameter.

ALPHA is COMPLEX

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, ka), where ka is. k when TRANS = 'N' or 'n', and is n otherwise. Before entry with TRANS = 'N' or 'n', the leading n by k part of the array A must contain the matrix A, otherwise the leading k by n part of the array A must contain the matrix A.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDA must be at least max(1, n), otherwise LDA must be at least max(1, k).

BETA Input parameter.

BETA is COMPLEX

On entry, BETA specifies the scalar beta.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array C must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of C is not referenced. On exit, the upper triangular part of the array C is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array C must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of C is not referenced. On exit, the lower triangular part of the array C is overwritten by the lower triangular part of the updated matrix.

LDC Input parameter.

LDC is INTEGER

On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max(1, n).

Related Information

For this routine in other precisions, please see *dsyrk*, *ssyrk* and *zsyrk*. It also exists with a native C interface as *cblas_csyrk*.

3.4.8 ctrmm

ctrmm performs one of the matrix-matrix operations

```
B := alpha*op( A )*B,   or   B := alpha*B*op( A )
```

where alpha is a scalar, B is an m by n matrix, A is a unit, or non-unit, upper or lower triangular matrix and op(A) is one of

```
op( A ) = A   or   op( A ) = A**T   or   op( A ) = A**H.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine ctrmm(SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB)
```

C specification:

```
#include "armpl.h"

void ctrmm_(const char *side, const char *uplo, const char *transa,
            const char *diag, const armpl_int_t *m, const armpl_int_t *n,
            const armpl_singlecomplex_t *alpha,
            const armpl_singlecomplex_t *a, const armpl_int_t *lda,
            armpl_singlecomplex_t *b, const armpl_int_t *ldb, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

On entry, SIDE specifies whether op(A) multiplies B from the left or right as follows:

SIDE = 'L' or 'l' B := alpha*op(A)*B.

SIDE = 'R' or 'r' B := alpha*B*op(A).

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the matrix A is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

TRANSA Input parameter.

TRANSA is CHARACTER*1

On entry, TRANSA specifies the form of op(A) to be used in the matrix multiplication as follows:

TRANSA = 'N' or 'n' op(A) = A.

TRANSA = 'T' or 't' op(A) = A^T .

TRANSA = 'C' or 'c' op(A) = A^H .

DIAG Input parameter.

DIAG is CHARACTER*1

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of B. M must be at least zero.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of B. N must be at least zero.

ALPHA Input parameter.

ALPHA is COMPLEX

On entry, ALPHA specifies the scalar alpha. When alpha is zero then A is not referenced and B need not be set before entry.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, k), where k is m. when SIDE = 'L' or 'l' and is n when SIDE = 'R' or 'r'. Before entry with UPLO = 'U' or 'u', the leading k by k upper triangular part of the array A must contain the upper triangular matrix and the strictly lower triangular part of A is not referenced. Before entry with UPLO = 'L' or 'l', the leading k by k lower triangular part of the array A must contain the lower triangular matrix and the strictly upper triangular part of A is not referenced. Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When SIDE = 'L' or 'l' then LDA must be at least max(1, m), when SIDE = 'R' or 'r' then LDA must be at least max(1, n).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, N).. Before entry, the leading m by n part of the array B must contain the matrix B, and on exit is overwritten by the transformed matrix.

LDB Input parameter.

LDB is INTEGER

On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. LDB must be at least max(1, m).

Related Information

For this routine in other precisions, please see *dtrmm*, *strmm* and *ztrmm*. It also exists with a native C interface as *cblas_ctrmm*.

3.4.9 ctrsm

ctrsm solves one of the matrix equations

```
op( A )*X = alpha*B,   or   X*op( A ) = alpha*B,
```

where alpha is a scalar, X and B are m by n matrices, A is a unit, or non-unit, upper or lower triangular matrix and op(A) is one of

```
op( A ) = A   or   op( A ) = A**T   or   op( A ) = A**H.
```

The matrix X is overwritten on B.

Syntax

Fortran specification:

```
use armpl_library

subroutine ctrsm(SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB)
```

C specification:

```
#include "armpl.h"

void ctrsm_(const char *side, const char *uplo, const char *transa,
            const char *diag, const armpl_int_t *m, const armpl_int_t *n,
            const armpl_singlecomplex_t *alpha,
            const armpl_singlecomplex_t *a, const armpl_int_t *lda,
            armpl_singlecomplex_t *b, const armpl_int_t *ldb, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

On entry, SIDE specifies whether op(A) appears on the left or right of X as follows:

SIDE = 'L' or 'l' op(A)*X = alpha*B.

SIDE = 'R' or 'r' X*op(A) = alpha*B.

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the matrix A is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

TRANSA Input parameter.

TRANSA is CHARACTER*1

On entry, TRANSA specifies the form of op(A) to be used in the matrix multiplication as follows:

TRANSA = 'N' or 'n' op(A) = A.

TRANSA = 'T' or 't' op(A) = A^T .

TRANSA = 'C' or 'c' op(A) = A^H .

DIAG Input parameter.

DIAG is CHARACTER*1

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of B. M must be at least zero.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of B. N must be at least zero.

ALPHA Input parameter.

ALPHA is COMPLEX

On entry, ALPHA specifies the scalar alpha. When alpha is zero then A is not referenced and B need not be set before entry.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, k),. where k is m when SIDE = 'L' or 'l' and k is n when SIDE = 'R' or 'r'. Before entry with UPLO = 'U' or 'u', the leading k by k upper triangular part of the array A must contain the upper triangular matrix and the strictly lower triangular part of A is not referenced. Before entry with UPLO = 'L' or 'l', the leading k by k lower triangular part of the array A must contain the lower triangular matrix and the strictly upper triangular part of A is not referenced. Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When SIDE = 'L' or 'l' then LDA must be at least max(1, m), when SIDE = 'R' or 'r' then LDA must be at least max(1, n).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, N). Before entry, the leading m by n part of the array B must contain the right-hand side matrix B, and on exit is overwritten by the solution matrix X.

LDB Input parameter.

LDB is INTEGER

On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. LDB must be at least max(1, m).

Related Information

For this routine in other precisions, please see *dtrsm*, *strsm* and *ztrsm*. It also exists with a native C interface as *cblas_ctrsm*.

3.4.10 dgemm

dgemm performs one of the matrix-matrix operations

```
C := alpha*op( A )*op( B ) + beta*C,
```

where op(X) is one of

```
op( X ) = X   or   op( X ) = X**T,
```

alpha and beta are scalars, and A, B and C are matrices, with op(A) an m by k matrix, op(B) a k by n matrix and C an m by n matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgemm(TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C,
                 LDC)
```

C specification:

```
#include "armpl.h"

void dgemm_(const char *transa, const char *transb, const armpl_int_t *m,
            const armpl_int_t *n, const armpl_int_t *k, const double *alpha,
            const double *a, const armpl_int_t *lda, const double *b,
            const armpl_int_t *ldb, const double *beta, double *c,
            const armpl_int_t *ldc, ... );
```

Parameters

TRANSA Input parameter.

TRANSA is CHARACTER*1

On entry, TRANSA specifies the form of op(A) to be used in the matrix multiplication as follows:

TRANSA = 'N' or 'n', op(A) = A.

TRANSA = 'T' or 't', op(A) = A^T .

TRANSA = 'C' or 'c', op(A) = A^T .

TRANSB Input parameter.

TRANSB is CHARACTER*1

On entry, TRANSB specifies the form of op(B) to be used in the matrix multiplication as follows:

TRANSB = 'N' or 'n', op(B) = B.

TRANSB = 'T' or 't', op(B) = B^T .

TRANSB = 'C' or 'c', op(B) = B^T .

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of the matrix op(A) and of the matrix C. M must be at least zero.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix op(B) and the number of columns of the matrix C. N must be at least zero.

K Input parameter.

K is INTEGER

On entry, K specifies the number of columns of the matrix op(A) and the number of rows of the matrix op(B). K must be at least zero.

ALPHA Input parameter.

ALPHA is DOUBLE PRECISION.

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, ka), where ka is. k when TRANSA = 'N' or 'n', and is m otherwise. Before entry with TRANSA = 'N' or 'n', the leading m by k part of the array A must contain the matrix A, otherwise the leading k by m part of the array A must contain the matrix A.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANSA = 'N' or 'n' then LDA must be at least max(1, m), otherwise LDA must be at least max(1, k).

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, kb), where kb is. n when TRANSB = 'N' or 'n', and is k otherwise. Before entry with TRANSB = 'N' or 'n', the leading k by n part of the array B must contain the matrix B, otherwise the leading n by k part of the array B must contain the matrix B.

LDB Input parameter.

LDB is INTEGER

On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. When TRANSB = 'N' or 'n' then LDB must be at least max(1, k), otherwise LDB must be at least max(1, n).

BETA Input parameter.

BETA is DOUBLE PRECISION.

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then C need not be set on input.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, N). Before entry, the leading m by n part of the array C must contain the matrix C, except when beta is zero, in which case C need not be set on entry. On exit, the array C is overwritten by the m by n matrix (alpha*op(A)*op(B) + beta* C).

LDC Input parameter.

LDC is INTEGER

On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max(1, m).

Related Information

For this routine in other precisions, please see *cgemm*, *hgemm*, *sgemm* and *zgemm*. It also exists with a native C interface as *cblas_dgemm*.

3.4.11 dsymm

dsymm performs one of the matrix-matrix operations

```
C := alpha*A*B + beta*C,
```

or

```
C := alpha*B*A + beta*C,
```

where alpha and beta are scalars, A is a symmetric matrix and B and C are m by n matrices.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsymm(SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
```

C specification:

```
#include "armpl.h"

void dsymm_(const char *side, const char *uplo, const armpl_int_t *m,
            const armpl_int_t *n, const double *alpha, const double *a,
            const armpl_int_t *lda, const double *b, const armpl_int_t *ldb,
            const double *beta, double *c, const armpl_int_t *ldc, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

On entry, SIDE specifies whether the symmetric matrix A appears on the left or right in the operation as follows:

SIDE = 'L' or 'l' C := alpha*A* B + beta*C,

SIDE = 'R' or 'r' C := alpha*B* A + beta*C,

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the symmetric matrix A is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of the symmetric matrix is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of the symmetric matrix is to be referenced.

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of the matrix C. M must be at least zero.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix C. N must be at least zero.

ALPHA Input parameter.

ALPHA is DOUBLE PRECISION.

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, ka), where ka is. m when SIDE = 'L' or 'l' and is n otherwise. Before entry with SIDE = 'L' or 'l', the m by m part of the array A must contain the symmetric matrix, such that when UPLO = 'U' or 'u', the leading m by m upper triangular part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not referenced, and when UPLO = 'L' or 'l', the leading m by m lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced. Before entry with SIDE = 'R' or 'r', the n by n part of the array A must contain the symmetric matrix, such that when UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not referenced, and when UPLO = 'L' or 'l',

the leading n by n lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When SIDE = 'L' or 'l' then LDA must be at least max(1, m), otherwise LDA must be at least max(1, n).

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). Before entry, the leading m by n part of the array B must contain the matrix B.

LDB Input parameter.

LDB is INTEGER

On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. LDB must be at least max(1, m).

BETA Input parameter.

BETA is DOUBLE PRECISION.

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then C need not be set on input.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, N). Before entry, the leading m by n part of the array C must contain the matrix C, except when beta is zero, in which case C need not be set on entry. On exit, the array C is overwritten by the m by n updated matrix.

LDC Input parameter.

LDC is INTEGER

On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max(1, m).

Related Information

For this routine in other precisions, please see *csymm*, *ssymm* and *zsymm*. It also exists with a native C interface as *cblas_dsymm*.

3.4.12 dsyr2k

dsyr2k performs one of the symmetric rank 2k operations

```
C := alpha*A*B**T + alpha*B*A**T + beta*C,
```

or

```
C := alpha*A**T*B + alpha*B**T*A + beta*C,
```

where alpha and beta are scalars, C is an n by n symmetric matrix and A and B are n by k matrices in the first case and k by n matrices in the second case.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsyr2k(UPLO, TRANS, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
```

C specification:

```
#include "armpl.h"

void dsyr2k_(const char *uplo, const char *trans, const armpl_int_t *n,
             const armpl_int_t *k, const double *alpha, const double *a,
             const armpl_int_t *lda, const double *b, const armpl_int_t *ldb,
             const double *beta, double *c, const armpl_int_t *ldc, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the array C is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of C is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of C is to be referenced.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' C := alpha*A*B^T + alpha*B*A^T + beta*C.

TRANS = 'T' or 't' C := alpha*A^T * B + alpha*B^T * A + beta*C.

TRANS = 'C' or 'c' C := alpha*A^T * B + alpha*B^T * A + beta*C.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix C. N must be at least zero.

K Input parameter.

K is INTEGER

On entry with TRANS = 'N' or 'n', K specifies the number of columns of the matrices A and B, and on entry with TRANS = 'T' or 't' or 'C' or 'c', K specifies the number of rows of the matrices A and B. K must be at least zero.

ALPHA Input parameter.

ALPHA is DOUBLE PRECISION.

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, ka), where ka is. k when TRANS = 'N' or 'n', and is n otherwise. Before entry with TRANS = 'N' or 'n', the leading n by k part of the array A must contain the matrix A, otherwise the leading k by n part of the array A must contain the matrix A.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDA must be at least max(1, n), otherwise LDA must be at least max(1, k).

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, kb), where kb is. k when TRANS = 'N' or 'n', and is n otherwise. Before entry with TRANS = 'N' or 'n', the leading n by k part of the array B must contain the matrix B, otherwise the leading k by n part of the array B must contain the matrix B.

LDB Input parameter.

LDB is INTEGER

On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDB must be at least max(1, n), otherwise LDB must be at least max(1, k).

BETA Input parameter.

BETA is DOUBLE PRECISION.

On entry, BETA specifies the scalar beta.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, N). Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array C must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of C is not referenced. On exit, the upper triangular part of the array C is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array C must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of C is not referenced. On exit, the lower triangular part of the array C is overwritten by the lower triangular part of the updated matrix.

LDC Input parameter.

LDC is INTEGER

On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max(1, n).

Related Information

For this routine in other precisions, please see *csyr2k*, *ssyr2k* and *zsyr2k*. It also exists with a native C interface as *cblas_dsyr2k*.

3.4.13 dsyrk

dsyrk performs one of the symmetric rank k operations

```
C := alpha*A*A**T + beta*C,
```

or

```
C := alpha*A**T*A + beta*C,
```

where alpha and beta are scalars, C is an n by n symmetric matrix and A is an n by k matrix in the first case and a k by n matrix in the second case.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsyrk(UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C, LDC)
```

C specification:

```
#include "armpl.h"

void dsyrk_(const char *uplo, const char *trans, const armpl_int_t *n,
            const armpl_int_t *k, const double *alpha, const double *a,
            const armpl_int_t *lda, const double *beta, double *c,
            const armpl_int_t *ldc, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the array C is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of C is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of C is to be referenced.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' C := alpha*A*A^T + beta*C.

TRANS = 'T' or 't' C := alpha*A^T * A + beta*C.

TRANS = 'C' or 'c' C := alpha*A^T * A + beta*C.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix C. N must be at least zero.

K Input parameter.

K is INTEGER

On entry with TRANS = 'N' or 'n', K specifies the number of columns of the matrix A, and on entry with TRANS = 'T' or 't' or 'C' or 'c', K specifies the number of rows of the matrix A. K must be at least zero.

ALPHA Input parameter.

ALPHA is DOUBLE PRECISION.

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, ka), where ka is. k when TRANS = 'N' or 'n', and is n otherwise. Before entry with TRANS = 'N' or 'n', the leading n by k part of the array A must contain the matrix A, otherwise the leading k by n part of the array A must contain the matrix A.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDA must be at least max(1, n), otherwise LDA must be at least max(1, k).

BETA Input parameter.

BETA is DOUBLE PRECISION.

On entry, BETA specifies the scalar beta.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, N). Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array C must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of C is not referenced. On exit, the upper triangular part of the array C is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array C must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of C is not referenced. On exit, the lower triangular part of the array C is overwritten by the lower triangular part of the updated matrix.

LDC Input parameter.

LDC is INTEGER

On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max(1, n).

Related Information

For this routine in other precisions, please see *csyrk*, *ssyrk* and *zsyrk*. It also exists with a native C interface as *cblas_dsyrk*.

3.4.14 dtrmm

dtrmm performs one of the matrix-matrix operations

```
B := alpha*op( A )*B,   or   B := alpha*B*op( A ),
```

where alpha is a scalar, B is an m by n matrix, A is a unit, or non-unit, upper or lower triangular matrix and op(A) is one of

```
op( A ) = A   or   op( A ) = A**T.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dtrmm(SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB)
```

C specification:

```
#include "armpl.h"

void dtrmm_(const char *side, const char *uplo, const char *transa,
            const char *diag, const armpl_int_t *m, const armpl_int_t *n,
```

(continues on next page)

(continued from previous page)

```
            const double *alpha, const double *a, const armpl_int_t *lda,
            double *b, const armpl_int_t *ldb, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

On entry, SIDE specifies whether op(A) multiplies B from the left or right as follows:

SIDE = 'L' or 'l' B := alpha*op(A)*B.

SIDE = 'R' or 'r' B := alpha*B*op(A).

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the matrix A is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

TRANSA Input parameter.

TRANSA is CHARACTER*1

On entry, TRANSA specifies the form of op(A) to be used in the matrix multiplication as follows:

TRANSA = 'N' or 'n' op(A) = A.

TRANSA = 'T' or 't' op(A) = A^T .

TRANSA = 'C' or 'c' op(A) = A^T .

DIAG Input parameter.

DIAG is CHARACTER*1

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of B. M must be at least zero.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of B. N must be at least zero.

ALPHA Input parameter.

ALPHA is DOUBLE PRECISION.

On entry, ALPHA specifies the scalar alpha. When alpha is zero then A is not referenced and B need not be set before entry.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, k), where k is m. when SIDE = 'L' or 'l' and is n when SIDE = 'R' or 'r'. Before entry with UPLO = 'U' or 'u', the leading k by k upper triangular part of the array A must contain the

upper triangular matrix and the strictly lower triangular part of A is not referenced. Before entry with UPLO = 'L' or 'l', the leading k by k lower triangular part of the array A must contain the lower triangular matrix and the strictly upper triangular part of A is not referenced. Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When SIDE = 'L' or 'l' then LDA must be at least max(1, m), when SIDE = 'R' or 'r' then LDA must be at least max(1, n).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). Before entry, the leading m by n part of the array B must contain the matrix B, and on exit is overwritten by the transformed matrix.

LDB Input parameter.

LDB is INTEGER

On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. LDB must be at least max(1, m).

Related Information

For this routine in other precisions, please see *ctrmm*, *strmm* and *ztrmm*. It also exists with a native C interface as *cblas_dtrmm*.

3.4.15 dtrsm

dtrsm solves one of the matrix equations

```
op( A )*X = alpha*B,   or   X*op( A ) = alpha*B,
```

where alpha is a scalar, X and B are m by n matrices, A is a unit, or non-unit, upper or lower triangular matrix and op(A) is one of

```
op( A ) = A   or   op( A ) = A**T.
```

The matrix X is overwritten on B.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtrsm(SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB)
```

C specification:

```
#include "armpl.h"

void dtrsm_(const char *side, const char *uplo, const char *transa,
            const char *diag, const armpl_int_t *m, const armpl_int_t *n,
            const double *alpha, const double *a, const armpl_int_t *lda,
            double *b, const armpl_int_t *ldb, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

On entry, SIDE specifies whether op(A) appears on the left or right of X as follows:

SIDE = 'L' or 'l' op(A)*X = alpha*B.

SIDE = 'R' or 'r' X*op(A) = alpha*B.

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the matrix A is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

TRANSA Input parameter.

TRANSA is CHARACTER*1

On entry, TRANSA specifies the form of op(A) to be used in the matrix multiplication as follows:

TRANSA = 'N' or 'n' op(A) = A.

TRANSA = 'T' or 't' op(A) = A^T .

TRANSA = 'C' or 'c' op(A) = A^T .

DIAG Input parameter.

DIAG is CHARACTER*1

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of B. M must be at least zero.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of B. N must be at least zero.

ALPHA Input parameter.

ALPHA is DOUBLE PRECISION.

On entry, ALPHA specifies the scalar alpha. When alpha is zero then A is not referenced and B need not be set before entry.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, k),. where k is m when SIDE = 'L' or 'l' and k is n when SIDE = 'R' or 'r'. Before entry with UPLO = 'U' or 'u', the leading k by k upper triangular part of the array A must contain the upper triangular matrix and the strictly lower triangular part of A is not referenced. Before entry with UPLO = 'L' or 'l', the leading k by k lower triangular part of the array A must contain the lower triangular matrix and the strictly upper triangular part of A is not referenced. Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When SIDE = 'L' or 'l' then LDA must be at least max(1, m), when SIDE = 'R' or 'r' then LDA must be at least max(1, n).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). Before entry, the leading m by n part of the array B must contain the right-hand side matrix B, and on exit is overwritten by the solution matrix X.

LDB Input parameter.

LDB is INTEGER

On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. LDB must be at least max(1, m).

Related Information

For this routine in other precisions, please see *ctrsm*, *strsm* and *ztrsm*. It also exists with a native C interface as *cblas_dtrsm*.

3.4.16 sgemm

sgemm performs one of the matrix-matrix operations

```
C := alpha*op( A )*op( B ) + beta*C,
```

where op(X) is one of

```
op( X ) = X   or   op( X ) = X**T,
```

alpha and beta are scalars, and A, B and C are matrices, with op(A) an m by k matrix, op(B) a k by n matrix and C an m by n matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgemm(TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C,
                 LDC)
```

C specification:

```
#include "armpl.h"

void sgemm_(const char *transa, const char *transb, const armpl_int_t *m,
            const armpl_int_t *n, const armpl_int_t *k, const float *alpha,
            const float *a, const armpl_int_t *lda, const float *b,
            const armpl_int_t *ldb, const float *beta, float *c,
            const armpl_int_t *ldc, ... );
```

Parameters

TRANSA Input parameter.

TRANSA is CHARACTER*1

On entry, TRANSA specifies the form of op(A) to be used in the matrix multiplication as follows:

TRANSA = 'N' or 'n', op(A) = A.

TRANSA = 'T' or 't', op(A) = A^T .

TRANSA = 'C' or 'c', op(A) = A^T .

TRANSB Input parameter.

TRANSB is CHARACTER*1

On entry, TRANSB specifies the form of op(B) to be used in the matrix multiplication as follows:

TRANSB = 'N' or 'n', op(B) = B.

TRANSB = 'T' or 't', op(B) = B^T .

TRANSB = 'C' or 'c', op(B) = B^T .

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of the matrix op(A) and of the matrix C. M must be at least zero.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix op(B) and the number of columns of the matrix C. N must be at least zero.

K Input parameter.

K is INTEGER

On entry, K specifies the number of columns of the matrix op(A) and the number of rows of the matrix op(B). K must be at least zero.

ALPHA Input parameter.

ALPHA is REAL

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is REAL

A is an array, dimension (LDA, ka), where ka is. k when TRANSA = 'N' or 'n', and is m otherwise. Before entry with TRANSA = 'N' or 'n', the leading m by k part of the array A must contain the matrix A, otherwise the leading k by m part of the array A must contain the matrix A.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANSA = 'N' or 'n' then LDA must be at least max(1, m), otherwise LDA must be at least max(1, k).

B Input parameter.

B is REAL

B is an array, dimension (LDB, kb), where kb is. n when TRANSB = 'N' or 'n', and is k otherwise. Before entry with TRANSB = 'N' or 'n', the leading k by n part of the array B must contain the matrix B, otherwise the leading n by k part of the array B must contain the matrix B.

LDB Input parameter.

LDB is INTEGER

On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. When TRANSB = 'N' or 'n' then LDB must be at least max(1, k), otherwise LDB must be at least max(1, n).

BETA Input parameter.

BETA is REAL

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then C need not be set on input.

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, N). Before entry, the leading m by n part of the array C must contain the matrix C, except when beta is zero, in which case C need not be set on entry. On exit, the array C is overwritten by the m by n matrix (alpha*op(A)*op(B) + beta* C).

LDC Input parameter.

LDC is INTEGER

On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max(1, m).

Related Information

For this routine in other precisions, please see *cgemm*, *dgemm*, *hgemm*, and *zgemm*. It also exists with a native C interface as *cblas_sgemm*.

3.4.17 ssymm

ssymm performs one of the matrix-matrix operations

```
C := alpha*A*B + beta*C,
```

or

```
C := alpha*B*A + beta*C,
```

where alpha and beta are scalars, A is a symmetric matrix and B and C are m by n matrices.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssymm(SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
```

C specification:

```
#include "armpl.h"

void ssymm_(const char *side, const char *uplo, const armpl_int_t *m,
            const armpl_int_t *n, const float *alpha, const float *a,
            const armpl_int_t *lda, const float *b, const armpl_int_t *ldb,
            const float *beta, float *c, const armpl_int_t *ldc, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

On entry, SIDE specifies whether the symmetric matrix A appears on the left or right in the operation as follows:

SIDE = 'L' or 'l' C := alpha*A* B + beta*C,

SIDE = 'R' or 'r' C := alpha*B* A + beta*C,

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the symmetric matrix A is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of the symmetric matrix is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of the symmetric matrix is to be referenced.

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of the matrix C. M must be at least zero.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix C. N must be at least zero.

ALPHA Input parameter.

ALPHA is REAL

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is REAL

A is an array, dimension (LDA, ka), where ka is. m when SIDE = 'L' or 'l' and is n otherwise. Before entry with SIDE = 'L' or 'l', the m by m part of the array A must contain the symmetric matrix, such that when UPLO = 'U' or 'u', the leading m by m upper triangular part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not referenced, and when UPLO = 'L' or 'l', the leading m by m lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced. Before entry with SIDE = 'R' or 'r', the n by n part of the array A must contain the symmetric matrix, such that when UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not referenced, and when UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When SIDE = 'L' or 'l' then LDA must be at least max(1, m), otherwise LDA must be at least max(1, n).

B Input parameter.

B is REAL

B is an array, dimension (LDB, N). Before entry, the leading m by n part of the array B must contain the matrix B.

LDB Input parameter.

LDB is INTEGER

On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. LDB must be at least max(1, m).

BETA Input parameter.

BETA is REAL

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then C need not be set on input.

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, N). Before entry, the leading m by n part of the array C must contain the matrix C, except when beta is zero, in which case C need not be set on entry. On exit, the array C is overwritten by the m by n updated matrix.

LDC Input parameter.

LDC is INTEGER

On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max(1, m).

Related Information

For this routine in other precisions, please see *csymm*, *dsymm* and *zsymm*. It also exists with a native C interface as *cblas_ssymm*.

3.4.18 ssyr2k

ssyr2k performs one of the symmetric rank 2k operations

```
C := alpha*A*B**T + alpha*B*A**T + beta*C,
```

or

```
C := alpha*A**T*B + alpha*B**T*A + beta*C,
```

where alpha and beta are scalars, C is an n by n symmetric matrix and A and B are n by k matrices in the first case and k by n matrices in the second case.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssyr2k(UPLO, TRANS, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
```

C specification:

```
#include "armpl.h"

void ssyr2k_(const char *uplo, const char *trans, const armpl_int_t *n,
             const armpl_int_t *k, const float *alpha, const float *a,
             const armpl_int_t *lda, const float *b, const armpl_int_t *ldb,
             const float *beta, float *c, const armpl_int_t *ldc, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the array C is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of C is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of C is to be referenced.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' C := alpha*A*B^T + alpha*B*A^T + beta*C.

TRANS = 'T' or 't' C := alpha*A^T * B + alpha*B^T * A + beta*C.

TRANS = 'C' or 'c' C := alpha*A^T * B + alpha*B^T * A + beta*C.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix C. N must be at least zero.

K Input parameter.

K is INTEGER

On entry with TRANS = 'N' or 'n', K specifies the number of columns of the matrices A and B, and on entry with TRANS = 'T' or 't' or 'C' or 'c', K specifies the number of rows of the matrices A and B. K must be at least zero.

ALPHA Input parameter.

ALPHA is REAL

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is REAL

A is an array, dimension (LDA, ka), where ka is. k when TRANS = 'N' or 'n', and is n otherwise. Before entry with TRANS = 'N' or 'n', the leading n by k part of the array A must contain the matrix A, otherwise the leading k by n part of the array A must contain the matrix A.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDA must be at least max(1, n), otherwise LDA must be at least max(1, k).

B Input parameter.

B is REAL

B is an array, dimension (LDB, kb), where kb is. k when TRANS = 'N' or 'n', and is n otherwise. Before entry with TRANS = 'N' or 'n', the leading n by k part of the array B must contain the matrix B, otherwise the leading k by n part of the array B must contain the matrix B.

LDB Input parameter.

LDB is INTEGER

On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDB must be at least max(1, n), otherwise LDB must be at least max(1, k).

BETA Input parameter.

BETA is REAL

On entry, BETA specifies the scalar beta.

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, N). Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array C must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of C is not referenced. On exit, the upper triangular part of the array C is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array C must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of C is not referenced. On exit, the lower triangular part of the array C is overwritten by the lower triangular part of the updated matrix.

LDC Input parameter.

LDC is INTEGER

On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max(1, n).

Related Information

For this routine in other precisions, please see *csyr2k*, *dsyr2k* and *zsyr2k*. It also exists with a native C interface as *cblas_ssyr2k*.

3.4.19 ssyrk

ssyrk performs one of the symmetric rank k operations

```
C := alpha*A*A**T + beta*C,
```

or

```
C := alpha*A**T*A + beta*C,
```

where alpha and beta are scalars, C is an n by n symmetric matrix and A is an n by k matrix in the first case and a k by n matrix in the second case.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssyrk(UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C, LDC)
```

C specification:

```
#include "armpl.h"

void ssyrk_(const char *uplo, const char *trans, const armpl_int_t *n,
            const armpl_int_t *k, const float *alpha, const float *a,
            const armpl_int_t *lda, const float *beta, float *c,
            const armpl_int_t *ldc, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the array C is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of C is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of C is to be referenced.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' C := alpha*A*A^T + beta*C.

TRANS = 'T' or 't' C := alpha*A^T * A + beta*C.

TRANS = 'C' or 'c' C := alpha*A^T * A + beta*C.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix C. N must be at least zero.

K Input parameter.

K is INTEGER

On entry with TRANS = 'N' or 'n', K specifies the number of columns of the matrix A, and on entry with TRANS = 'T' or 't' or 'C' or 'c', K specifies the number of rows of the matrix A. K must be at least zero.

ALPHA Input parameter.

ALPHA is REAL

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is REAL

A is an array, dimension (LDA, ka), where ka is. k when TRANS = 'N' or 'n', and is n otherwise. Before entry with TRANS = 'N' or 'n', the leading n by k part of the array A must contain the matrix A, otherwise the leading k by n part of the array A must contain the matrix A.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDA must be at least max(1, n), otherwise LDA must be at least max(1, k).

BETA Input parameter.

BETA is REAL

On entry, BETA specifies the scalar beta.

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, N). Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array C must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of C is not referenced. On exit, the upper triangular part of the array C is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array C must contain the lower triangular part of the symmetric matrix and the strictly

upper triangular part of C is not referenced. On exit, the lower triangular part of the array C is overwritten by the lower triangular part of the updated matrix.

LDC Input parameter.

LDC is INTEGER

On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max(1, n).

Related Information

For this routine in other precisions, please see *csyrk*, *dsyrk* and *zsyrk*. It also exists with a native C interface as *cblas_ssyrk*.

3.4.20 strmm

strmm performs one of the matrix-matrix operations

```
B := alpha*op( A )*B,   or   B := alpha*B*op( A ),
```

where alpha is a scalar, B is an m by n matrix, A is a unit, or non-unit, upper or lower triangular matrix and op(A) is one of

```
op( A ) = A   or   op( A ) = A**T.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine strmm(SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB)
```

C specification:

```
#include "armpl.h"

void strmm_(const char *side, const char *uplo, const char *transa,
            const char *diag, const armpl_int_t *m, const armpl_int_t *n,
            const float *alpha, const float *a, const armpl_int_t *lda,
            float *b, const armpl_int_t *ldb, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

On entry, SIDE specifies whether op(A) multiplies B from the left or right as follows:

SIDE = 'L' or 'l' B := alpha*op(A)*B.

SIDE = 'R' or 'r' B := alpha*B*op(A).

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the matrix A is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

TRANSA Input parameter.

TRANSA is CHARACTER*1

On entry, TRANSA specifies the form of op(A) to be used in the matrix multiplication as follows:

TRANSA = 'N' or 'n' op(A) = A.

TRANSA = 'T' or 't' op(A) = A^T .

TRANSA = 'C' or 'c' op(A) = A^T .

DIAG Input parameter.

DIAG is CHARACTER*1

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of B. M must be at least zero.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of B. N must be at least zero.

ALPHA Input parameter.

ALPHA is REAL

On entry, ALPHA specifies the scalar alpha. When alpha is zero then A is not referenced and B need not be set before entry.

A Input parameter.

A is REAL

A is an array, dimension (LDA, k), where k is m. when SIDE = 'L' or 'l' and is n when SIDE = 'R' or 'r'. Before entry with UPLO = 'U' or 'u', the leading k by k upper triangular part of the array A must contain the upper triangular matrix and the strictly lower triangular part of A is not referenced. Before entry with UPLO = 'L' or 'l', the leading k by k lower triangular part of the array A must contain the lower triangular matrix and the strictly upper triangular part of A is not referenced. Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When SIDE = 'L' or 'l' then LDA must be at least max(1, m), when SIDE = 'R' or 'r' then LDA must be at least max(1, n).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, N). Before entry, the leading m by n part of the array B must contain the matrix B, and on exit is overwritten by the transformed matrix.

LDB Input parameter.

LDB is INTEGER

On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. LDB must be at least max(1, m).

Related Information

For this routine in other precisions, please see *ctrmm*, *dtrmm* and *ztrmm*. It also exists with a native C interface as *cblas_strmm*.

3.4.21 strsm

strsm solves one of the matrix equations

```
op( A )*X = alpha*B,   or   X*op( A ) = alpha*B,
```

where alpha is a scalar, X and B are m by n matrices, A is a unit, or non-unit, upper or lower triangular matrix and op(A) is one of

```
op( A ) = A   or   op( A ) = A**T.
```

The matrix X is overwritten on B.

Syntax

Fortran specification:

```
use armpl_library

subroutine strsm(SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB)
```

C specification:

```
#include "armpl.h"

void strsm_(const char *side, const char *uplo, const char *transa,
            const char *diag, const armpl_int_t *m, const armpl_int_t *n,
            const float *alpha, const float *a, const armpl_int_t *lda,
            float *b, const armpl_int_t *ldb, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

On entry, SIDE specifies whether op(A) appears on the left or right of X as follows:

SIDE = 'L' or 'l' op(A)*X = alpha*B.

SIDE = 'R' or 'r' X*op(A) = alpha*B.

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the matrix A is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

TRANSA Input parameter.

TRANSA is CHARACTER*1

On entry, TRANSA specifies the form of op(A) to be used in the matrix multiplication as follows:

TRANSA = 'N' or 'n' op(A) = A.

TRANSA = 'T' or 't' op(A) = A^T .

TRANSA = 'C' or 'c' op(A) = A^T .

DIAG Input parameter.

DIAG is CHARACTER*1

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of B. M must be at least zero.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of B. N must be at least zero.

ALPHA Input parameter.

ALPHA is REAL

On entry, ALPHA specifies the scalar alpha. When alpha is zero then A is not referenced and B need not be set before entry.

A Input parameter.

A is REAL

A is an array, dimension (LDA, k),. where k is m when SIDE = 'L' or 'l' and k is n when SIDE = 'R' or 'r'. Before entry with UPLO = 'U' or 'u', the leading k by k upper triangular part of the array A must contain the upper triangular matrix and the strictly lower triangular part of A is not referenced. Before entry with UPLO = 'L' or 'l', the leading k by k lower triangular part of the array A must contain the lower triangular matrix and the strictly upper triangular part of A is not referenced. Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When SIDE = 'L' or 'l' then LDA must be at least max(1, m), when SIDE = 'R' or 'r' then LDA must be at least max(1, n).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, N). Before entry, the leading m by n part of the array B must contain the right-hand side matrix B, and on exit is overwritten by the solution matrix X.

LDB Input parameter.

LDB is INTEGER

On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. LDB must be at least max(1, m).

Related Information

For this routine in other precisions, please see *ctrsm*, *dtrsm* and *ztrsm*. It also exists with a native C interface as *cblas_strsm*.

3.4.22 zgemm

zgemm performs one of the matrix-matrix operations

```
C := alpha*op( A )*op( B ) + beta*C,
```

where op(X) is one of

```
op( X ) = X   or   op( X ) = X**T   or   op( X ) = X**H,
```

alpha and beta are scalars, and A, B and C are matrices, with op(A) an m by k matrix, op(B) a k by n matrix and C an m by n matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgemm(TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C,
                 LDC)
```

C specification:

```
#include "armpl.h"

void zgemm_(const char *transa, const char *transb, const armpl_int_t *m,
            const armpl_int_t *n, const armpl_int_t *k,
            const armpl_doublecomplex_t *alpha,
            const armpl_doublecomplex_t *a, const armpl_int_t *lda,
            const armpl_doublecomplex_t *b, const armpl_int_t *ldb,
            const armpl_doublecomplex_t *beta, armpl_doublecomplex_t *c,
            const armpl_int_t *ldc, ... );
```

Parameters

TRANSA Input parameter.

TRANSA is CHARACTER*1

On entry, TRANSA specifies the form of op(A) to be used in the matrix multiplication as follows:

TRANSA = 'N' or 'n', op(A) = A.

TRANSA = 'T' or 't', op(A) = A^T .

TRANSA = 'C' or 'c', op(A) = A^H .

TRANSB Input parameter.

TRANSB is CHARACTER*1

On entry, TRANSB specifies the form of op(B) to be used in the matrix multiplication as follows:

TRANSB = 'N' or 'n', op(B) = B.

TRANSB = 'T' or 't', op(B) = B^T .

TRANSB = 'C' or 'c', op(B) = B^H .

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of the matrix op(A) and of the matrix C. M must be at least zero.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix op(B) and the number of columns of the matrix C. N must be at least zero.

K Input parameter.

K is INTEGER

On entry, K specifies the number of columns of the matrix op(A) and the number of rows of the matrix op(B). K must be at least zero.

ALPHA Input parameter.

ALPHA is COMPLEX*16

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, ka), where ka is. k when TRANSA = 'N' or 'n', and is m otherwise. Before entry with TRANSA = 'N' or 'n', the leading m by k part of the array A must contain the matrix A, otherwise the leading k by m part of the array A must contain the matrix A.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANSA = 'N' or 'n' then LDA must be at least max(1, m), otherwise LDA must be at least max(1, k).

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, kb), where kb is. n when TRANSB = 'N' or 'n', and is k otherwise. Before entry with TRANSB = 'N' or 'n', the leading k by n part of the array B must contain the matrix B, otherwise the leading n by k part of the array B must contain the matrix B.

LDB Input parameter.

LDB is INTEGER

On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. When TRANSB = 'N' or 'n' then LDB must be at least max(1, k), otherwise LDB must be at least max(1, n).

BETA Input parameter.

BETA is COMPLEX*16

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then C need not be set on input.

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). Before entry, the leading m by n part of the array C must contain the matrix C, except when beta is zero, in which case C need not be set on entry. On exit, the array C is overwritten by the m by n matrix (alpha*op(A)*op(B) + beta* C).

LDC Input parameter.

LDC is INTEGER

On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max(1, m).

Related Information

For this routine in other precisions, please see *cgemm*, *dgemm*, *hgemm*, and *sgemm*. It also exists with a native C interface as *cblas_zgemm*.

3.4.23 zhemm

zhemm performs one of the matrix-matrix operations

```
C := alpha*A*B + beta*C,
```

or

```
C := alpha*B*A + beta*C,
```

where alpha and beta are scalars, A is an hermitian matrix and B and C are m by n matrices.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhemm(SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
```

C specification:

```
#include "armpl.h"

void zhemm_(const char *side, const char *uplo, const armpl_int_t *m,
            const armpl_int_t *n, const armpl_doublecomplex_t *alpha,
            const armpl_doublecomplex_t *a, const armpl_int_t *lda,
            const armpl_doublecomplex_t *b, const armpl_int_t *ldb,
            const armpl_doublecomplex_t *beta, armpl_doublecomplex_t *c,
            const armpl_int_t *ldc, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

On entry, SIDE specifies whether the hermitian matrix A appears on the left or right in the operation as follows:

SIDE = 'L' or 'l' C := alpha*A* B + beta*C,

SIDE = 'R' or 'r' C := alpha*B* A + beta*C,

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the hermitian matrix A is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of the hermitian matrix is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of the hermitian matrix is to be referenced.

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of the matrix C. M must be at least zero.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix C. N must be at least zero.

ALPHA Input parameter.

ALPHA is COMPLEX*16

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, ka), where ka is. m when SIDE = 'L' or 'l' and is n otherwise. Before entry with SIDE = 'L' or 'l', the m by m part of the array A must contain the hermitian matrix, such that when UPLO = 'U' or 'u', the leading m by m upper triangular part of the array A must contain the upper triangular part of the hermitian matrix and the strictly lower triangular part of A is not referenced, and when UPLO = 'L' or 'l', the leading m by m lower triangular part of the array A must contain the lower triangular part of the hermitian matrix and the strictly upper triangular part of A is not referenced. Before entry with SIDE = 'R' or 'r', the n by n part of the array A must contain the hermitian matrix, such that when UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the hermitian matrix and the strictly lower triangular part of A is not referenced, and when UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the hermitian matrix and the strictly upper triangular part of A is not referenced. Note that the imaginary parts of the diagonal elements need not be set, they are assumed to be zero.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When SIDE = 'L' or 'l' then LDA must be at least max(1, m), otherwise LDA must be at least max(1, n).

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). Before entry, the leading m by n part of the array B must contain the matrix B.

LDB Input parameter.

LDB is INTEGER

On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. LDB must be at least max(1, m).

BETA Input parameter.

BETA is COMPLEX*16

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then C need not be set on input.

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). Before entry, the leading m by n part of the array C must contain the matrix C, except when beta is zero, in which case C need not be set on entry. On exit, the array C is overwritten by the m by n updated matrix.

LDC Input parameter.

LDC is INTEGER

On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max(1, m).

Related Information

For this routine in other precisions, please see *chemm*. It also exists with a native C interface as *cblas_zhemm*.

3.4.24 zher2k

zher2k performs one of the hermitian rank 2k operations

```
C := alpha*A*B**H + conjg( alpha )*B*A**H + beta*C,
```

or

```
C := alpha*A**H*B + conjg( alpha )*B**H*A + beta*C,
```

where alpha and beta are scalars with beta real, C is an n by n hermitian matrix and A and B are n by k matrices in the first case and k by n matrices in the second case.

Syntax

Fortran specification:

```
use armpl_library

subroutine zher2k(UPLO, TRANS, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
```

C specification:

```
#include "armpl.h"

void zher2k_(const char *uplo, const char *trans, const armpl_int_t *n,
             const armpl_int_t *k, const armpl_doublecomplex_t *alpha,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_doublecomplex_t *b, const armpl_int_t *ldb,
             const double *beta, armpl_doublecomplex_t *c,
             const armpl_int_t *ldc, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the array C is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of C is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of C is to be referenced.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' C := alpha*A*B^H + conjg(alpha)*B*A^H + beta*C.

TRANS = 'C' or 'c' C := alpha*A^H * B + conjg(alpha)*B^H * A + beta*C.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix C. N must be at least zero.

K Input parameter.

K is INTEGER

On entry with TRANS = 'N' or 'n', K specifies the number of columns of the matrices A and B, and on entry with TRANS = 'C' or 'c', K specifies the number of rows of the matrices A and B. K must be at least zero.

ALPHA Input parameter.

ALPHA is COMPLEX*16 .

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, ka), where ka is. k when TRANS = 'N' or 'n', and is n otherwise. Before entry with TRANS = 'N' or 'n', the leading n by k part of the array A must contain the matrix A, otherwise the leading k by n part of the array A must contain the matrix A.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDA must be at least max(1, n), otherwise LDA must be at least max(1, k).

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, kb), where kb is. k when TRANS = 'N' or 'n', and is n otherwise. Before entry with TRANS = 'N' or 'n', the leading n by k part of the array B must contain the matrix B, otherwise the leading k by n part of the array B must contain the matrix B.

LDB Input parameter.

LDB is INTEGER

On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDB must be at least max(1, n), otherwise LDB must be at least max(1, k). Unchanged on exit.

BETA Input parameter.

BETA is DOUBLE PRECISION .

On entry, BETA specifies the scalar beta.

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array C must contain the upper triangular part of the hermitian matrix and the strictly lower triangular part of C is not referenced. On exit, the upper triangular part of the array C is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array C must contain the lower triangular part of the hermitian matrix and the strictly

upper triangular part of C is not referenced. On exit, the lower triangular part of the array C is overwritten by the lower triangular part of the updated matrix. Note that the imaginary parts of the diagonal elements need not be set, they are assumed to be zero, and on exit they are set to zero.

LDC Input parameter.

LDC is INTEGER

On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max(1, n).

Related Information

For this routine in other precisions, please see *cher2k*. It also exists with a native C interface as *cblas_zher2k*.

3.4.25 zherk

zherk performs one of the hermitian rank k operations

```
C := alpha*A*A**H + beta*C,
```

or

```
C := alpha*A**H*A + beta*C,
```

where alpha and beta are real scalars, C is an n by n hermitian matrix and A is an n by k matrix in the first case and a k by n matrix in the second case.

Syntax

Fortran specification:

```
use armpl_library

subroutine zherk(UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C, LDC)
```

C specification:

```
#include "armpl.h"

void zherk_(const char *uplo, const char *trans, const armpl_int_t *n,
            const armpl_int_t *k, const double *alpha,
            const armpl_doublecomplex_t *a, const armpl_int_t *lda,
            const double *beta, armpl_doublecomplex_t *c,
            const armpl_int_t *ldc, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the array C is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of C is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of C is to be referenced.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' C := alpha*A*A^H + beta*C.

TRANS = 'C' or 'c' C := alpha*A^H * A + beta*C.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix C. N must be at least zero.

K Input parameter.

K is INTEGER

On entry with TRANS = 'N' or 'n', K specifies the number of columns of the matrix A, and on entry with TRANS = 'C' or 'c', K specifies the number of rows of the matrix A. K must be at least zero.

ALPHA Input parameter.

ALPHA is DOUBLE PRECISION .

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, ka), where ka is. k when TRANS = 'N' or 'n', and is n otherwise. Before entry with TRANS = 'N' or 'n', the leading n by k part of the array A must contain the matrix A, otherwise the leading k by n part of the array A must contain the matrix A.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDA must be at least max(1, n), otherwise LDA must be at least max(1, k).

BETA Input parameter.

BETA is DOUBLE PRECISION.

On entry, BETA specifies the scalar beta.

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array C must contain the upper triangular part of the hermitian matrix and the strictly lower triangular part of C is not referenced. On exit, the upper triangular part of the array C is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array C must contain the lower triangular part of the hermitian matrix and the strictly upper triangular part of C is not referenced. On exit, the lower triangular part of the array C is overwritten by the lower triangular part of the updated matrix. Note that the imaginary parts of the diagonal elements need not be set, they are assumed to be zero, and on exit they are set to zero.

LDC Input parameter.

LDC is INTEGER

On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max(1, n).

Related Information

For this routine in other precisions, please see *cherk*. It also exists with a native C interface as *cblas_zherk*.

3.4.26 zsymm

zsymm performs one of the matrix-matrix operations

```
C := alpha*A*B + beta*C,
```

or

```
C := alpha*B*A + beta*C,
```

where alpha and beta are scalars, A is a symmetric matrix and B and C are m by n matrices.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsymm(SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
```

C specification:

```
#include "armpl.h"

void zsymm_(const char *side, const char *uplo, const armpl_int_t *m,
            const armpl_int_t *n, const armpl_doublecomplex_t *alpha,
            const armpl_doublecomplex_t *a, const armpl_int_t *lda,
            const armpl_doublecomplex_t *b, const armpl_int_t *ldb,
            const armpl_doublecomplex_t *beta, armpl_doublecomplex_t *c,
            const armpl_int_t *ldc, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

On entry, SIDE specifies whether the symmetric matrix A appears on the left or right in the operation as follows:

SIDE = 'L' or 'l' C := alpha*A* B + beta*C,

SIDE = 'R' or 'r' C := alpha*B* A + beta*C,

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the symmetric matrix A is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of the symmetric matrix is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of the symmetric matrix is to be referenced.

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of the matrix C. M must be at least zero.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix C. N must be at least zero.

ALPHA Input parameter.

ALPHA is COMPLEX*16

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, ka), where ka is. m when SIDE = 'L' or 'l' and is n otherwise. Before entry with SIDE = 'L' or 'l', the m by m part of the array A must contain the symmetric matrix, such that when UPLO = 'U' or 'u', the leading m by m upper triangular part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not referenced, and when UPLO = 'L' or 'l', the leading m by m lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced. Before entry with SIDE = 'R' or 'r', the n by n part of the array A must contain the symmetric matrix, such that when UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not referenced, and when UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When SIDE = 'L' or 'l' then LDA must be at least max(1, m), otherwise LDA must be at least max(1, n).

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). Before entry, the leading m by n part of the array B must contain the matrix B.

LDB Input parameter.

LDB is INTEGER

On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. LDB must be at least max(1, m).

BETA Input parameter.

BETA is COMPLEX*16

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then C need not be set on input.

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). Before entry, the leading m by n part of the array C must contain the matrix C, except when beta is zero, in which case C need not be set on entry. On exit, the array C is overwritten by the m by n updated matrix.

LDC Input parameter.

LDC is INTEGER

On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max(1, m).

Related Information

For this routine in other precisions, please see *csymm*, *dsymm* and *ssymm*. It also exists with a native C interface as *cblas_zsymm*.

3.4.27 zsyr2k

`zsyr2k` performs one of the symmetric rank 2k operations

```
C := alpha*A*B**T + alpha*B*A**T + beta*C,
```

or

```
C := alpha*A**T*B + alpha*B**T*A + beta*C,
```

where alpha and beta are scalars, C is an n by n symmetric matrix and A and B are n by k matrices in the first case and k by n matrices in the second case.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsyr2k(UPLO, TRANS, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
```

C specification:

```
#include "armpl.h"

void zsyr2k_(const char *uplo, const char *trans, const armpl_int_t *n,
             const armpl_int_t *k, const armpl_doublecomplex_t *alpha,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_doublecomplex_t *b, const armpl_int_t *ldb,
             const armpl_doublecomplex_t *beta, armpl_doublecomplex_t *c,
             const armpl_int_t *ldc, ... );
```

Parameters

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

On entry, `UPLO` specifies whether the upper or lower triangular part of the array `C` is to be referenced as follows:

`UPLO` = 'U' or 'u' Only the upper triangular part of C is to be referenced.

`UPLO` = 'L' or 'l' Only the lower triangular part of C is to be referenced.

TRANS Input parameter.

`TRANS` is `CHARACTER*1`

On entry, `TRANS` specifies the operation to be performed as follows:

`TRANS` = 'N' or 'n' `C` := alpha*A*B^T + alpha*B*A^T + beta*C.

`TRANS` = 'T' or 't' `C` := alpha*A^T * `B` + alpha*B^T * `A` + beta*C.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix C. N must be at least zero.

K Input parameter.

K is INTEGER

On entry with TRANS = 'N' or 'n', K specifies the number of columns of the matrices A and B, and on entry with TRANS = 'T' or 't', K specifies the number of rows of the matrices A and B. K must be at least zero.

ALPHA Input parameter.

ALPHA is COMPLEX*16

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, ka), where ka is. k when TRANS = 'N' or 'n', and is n otherwise. Before entry with TRANS = 'N' or 'n', the leading n by k part of the array A must contain the matrix A, otherwise the leading k by n part of the array A must contain the matrix A.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDA must be at least max(1, n), otherwise LDA must be at least max(1, k).

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, kb), where kb is. k when TRANS = 'N' or 'n', and is n otherwise. Before entry with TRANS = 'N' or 'n', the leading n by k part of the array B must contain the matrix B, otherwise the leading k by n part of the array B must contain the matrix B.

LDB Input parameter.

LDB is INTEGER

On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDB must be at least max(1, n), otherwise LDB must be at least max(1, k).

BETA Input parameter.

BETA is COMPLEX*16

On entry, BETA specifies the scalar beta.

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array C must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of C is not referenced. On exit, the upper triangular part of the array C is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array C must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of C is not referenced. On exit, the lower triangular part of the array C is overwritten by the lower triangular part of the updated matrix.

LDC Input parameter.

LDC is INTEGER

On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max(1, n).

Related Information

For this routine in other precisions, please see *csyr2k*, *dsyr2k* and *ssyr2k*. It also exists with a native C interface as *cblas_zsyr2k*.

3.4.28 zsyrk

`zsyrk` performs one of the symmetric rank k operations

```
C := alpha*A*A**T + beta*C,
```

or

```
C := alpha*A**T*A + beta*C,
```

where alpha and beta are scalars, C is an n by n symmetric matrix and A is an n by k matrix in the first case and a k by n matrix in the second case.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsyrk(UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C, LDC)
```

C specification:

```
#include "armpl.h"

void zsyrk_(const char *uplo, const char *trans, const armpl_int_t *n,
            const armpl_int_t *k, const armpl_doublecomplex_t *alpha,
            const armpl_doublecomplex_t *a, const armpl_int_t *lda,
            const armpl_doublecomplex_t *beta, armpl_doublecomplex_t *c,
            const armpl_int_t *ldc, ... );
```

Parameters

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

On entry, `UPLO` specifies whether the upper or lower triangular part of the array `C` is to be referenced as follows:

`UPLO` = '`U`' or '`u`' Only the upper triangular part of C is to be referenced.

`UPLO` = '`L`' or '`l`' Only the lower triangular part of C is to be referenced.

TRANS Input parameter.

`TRANS` is `CHARACTER*1`

On entry, `TRANS` specifies the operation to be performed as follows:

`TRANS` = '`N`' or '`n`' `C` := alpha*A*A^T + beta*C.

`TRANS` = '`T`' or '`t`' `C` := alpha*A^T * `A` + beta*C.

N Input parameter.

`N` is `INTEGER`

On entry, N specifies the order of the matrix C. N must be at least zero.

K Input parameter.

K is INTEGER

On entry with TRANS = 'N' or 'n', K specifies the number of columns of the matrix A, and on entry with TRANS = 'T' or 't', K specifies the number of rows of the matrix A. K must be at least zero.

ALPHA Input parameter.

ALPHA is COMPLEX*16

On entry, ALPHA specifies the scalar alpha.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, ka), where ka is. k when TRANS = 'N' or 'n', and is n otherwise. Before entry with TRANS = 'N' or 'n', the leading n by k part of the array A must contain the matrix A, otherwise the leading k by n part of the array A must contain the matrix A.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDA must be at least max(1, n), otherwise LDA must be at least max(1, k).

BETA Input parameter.

BETA is COMPLEX*16

On entry, BETA specifies the scalar beta.

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array C must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of C is not referenced. On exit, the upper triangular part of the array C is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array C must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of C is not referenced. On exit, the lower triangular part of the array C is overwritten by the lower triangular part of the updated matrix.

LDC Input parameter.

LDC is INTEGER

On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max(1, n).

Related Information

For this routine in other precisions, please see *csyrk*, *dsyrk* and *ssyrk*. It also exists with a native C interface as *cblas_zsyrk*.

3.4.29 ztrmm

ztrmm performs one of the matrix-matrix operations

```
B := alpha*op( A )*B,   or   B := alpha*B*op( A )
```

where alpha is a scalar, B is an m by n matrix, A is a unit, or non-unit, upper or lower triangular matrix and op(A) is one of

```
op( A ) = A   or   op( A ) = A**T   or   op( A ) = A**H.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine ztrmm(SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB)
```

C specification:

```
#include "armpl.h"

void ztrmm_(const char *side, const char *uplo, const char *transa,
            const char *diag, const armpl_int_t *m, const armpl_int_t *n,
            const armpl_doublecomplex_t *alpha,
            const armpl_doublecomplex_t *a, const armpl_int_t *lda,
            armpl_doublecomplex_t *b, const armpl_int_t *ldb, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

On entry, SIDE specifies whether op(A) multiplies B from the left or right as follows:

SIDE = 'L' or 'l' B := alpha*op(A)*B.

SIDE = 'R' or 'r' B := alpha*B*op(A).

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the matrix A is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

TRANSA Input parameter.

TRANSA is CHARACTER*1

On entry, TRANSA specifies the form of op(A) to be used in the matrix multiplication as follows:

TRANSA = 'N' or 'n' op(A) = A.

TRANSA = 'T' or 't' op(A) = A^T .

TRANSA = 'C' or 'c' op(A) = A^H .

DIAG Input parameter.

DIAG is CHARACTER*1

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of B. M must be at least zero.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of B. N must be at least zero.

ALPHA Input parameter.

ALPHA is COMPLEX*16

On entry, ALPHA specifies the scalar alpha. When alpha is zero then A is not referenced and B need not be set before entry.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, k), where k is m. when SIDE = 'L' or 'l' and is n when SIDE = 'R' or 'r'. Before entry with UPLO = 'U' or 'u', the leading k by k upper triangular part of the array A must contain the upper triangular matrix and the strictly lower triangular part of A is not referenced. Before entry with UPLO = 'L' or 'l', the leading k by k lower triangular part of the array A must contain the lower triangular matrix and the strictly upper triangular part of A is not referenced. Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When SIDE = 'L' or 'l' then LDA must be at least max(1, m), when SIDE = 'R' or 'r' then LDA must be at least max(1, n).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N).. Before entry, the leading m by n part of the array B must contain the matrix B, and on exit is overwritten by the transformed matrix.

LDB Input parameter.

LDB is INTEGER

On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. LDB must be at least max(1, m).

Related Information

For this routine in other precisions, please see *ctrmm*, *dtrmm* and *strmm*. It also exists with a native C interface as *cblas_ztrmm*.

3.4.30 ztrsm

ztrsm solves one of the matrix equations

```
op( A )*X = alpha*B,   or   X*op( A ) = alpha*B,
```

where alpha is a scalar, X and B are m by n matrices, A is a unit, or non-unit, upper or lower triangular matrix and op(A) is one of

```
op( A ) = A   or   op( A ) = A**T   or   op( A ) = A**H.
```

The matrix X is overwritten on B.

Syntax

Fortran specification:

```
use armpl_library

subroutine ztrsm(SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB)
```

C specification:

```
#include "armpl.h"

void ztrsm_(const char *side, const char *uplo, const char *transa,
            const char *diag, const armpl_int_t *m, const armpl_int_t *n,
            const armpl_doublecomplex_t *alpha,
            const armpl_doublecomplex_t *a, const armpl_int_t *lda,
            armpl_doublecomplex_t *b, const armpl_int_t *ldb, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

On entry, SIDE specifies whether op(A) appears on the left or right of X as follows:

SIDE = 'L' or 'l' op(A)*X = alpha*B.

SIDE = 'R' or 'r' X*op(A) = alpha*B.

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the matrix A is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

TRANSA Input parameter.

TRANSA is CHARACTER*1

On entry, TRANSA specifies the form of op(A) to be used in the matrix multiplication as follows:

TRANSA = 'N' or 'n' op(A) = A.

TRANSA = 'T' or 't' op(A) = A^T .

TRANSA = 'C' or 'c' op(A) = A^H .

DIAG Input parameter.

DIAG is CHARACTER*1

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of B. M must be at least zero.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of B. N must be at least zero.

ALPHA Input parameter.

ALPHA is COMPLEX*16

On entry, ALPHA specifies the scalar alpha. When alpha is zero then A is not referenced and B need not be set before entry.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, k),. where k is m when SIDE = 'L' or 'l' and k is n when SIDE = 'R' or 'r'. Before entry with UPLO = 'U' or 'u', the leading k by k upper triangular part of the array A must contain the upper triangular matrix and the strictly lower triangular part of A is not referenced. Before entry with UPLO = 'L' or 'l', the leading k by k lower triangular part of the array A must contain the lower triangular matrix and the strictly upper triangular part of A is not referenced. Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When SIDE = 'L' or 'l' then LDA must be at least max(1, m), when SIDE = 'R' or 'r' then LDA must be at least max(1, n).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). Before entry, the leading m by n part of the array B must contain the right-hand side matrix B, and on exit is overwritten by the solution matrix X.

LDB Input parameter.

LDB is INTEGER

On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. LDB must be at least max(1, m).

Related Information

For this routine in other precisions, please see *ctrsm*, *dtrsm* and *strsm*. It also exists with a native C interface as *cblas_ztrsm*.

Level 3 BLAS routines are the computationally intensive routines, normally working on multiple dense matrices. All functions are detailed in armpl.h.

3.5 CBLAS functions

3.5.1 cblas_caxpby

Syntax

```
#include "armpl.h"

void cblas_caxpby(armpl_int_t N, const armpl_singlecomplex_t *alpha,
                  const armpl_singlecomplex_t *X, armpl_int_t incX,
```

(continues on next page)

(continued from previous page)

```
                    const armpl_singlecomplex_t *beta,
                    armpl_singlecomplex_t *Y, armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_caxpby*, *cblas_saxpy* and *cblas_zaxpby*. It also exists with a native Fortran interface as *caxpby*.

3.5.2 cblas_caxpy

Syntax

```
#include "armpl.h"

void cblas_caxpy(const armpl_int_t N, const armpl_singlecomplex_t*alpha,
                 const armpl_singlecomplex_t*X, const armpl_int_t incX,
                 armpl_singlecomplex_t*Y, const armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_daxpy*, *cblas_saxpy* and *cblas_zaxpy*. It also exists with a native Fortran interface as *caxpy*.

3.5.3 cblas_ccopy

Syntax

```
#include "armpl.h"

void cblas_ccopy(const armpl_int_t N, const armpl_singlecomplex_t*X,
                 const armpl_int_t incX, armpl_singlecomplex_t*Y,
                 const armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dcopy*, *cblas_scopy* and *cblas_zcopy*. It also exists with a native Fortran interface as *ccopy*.

3.5.4 cblas_cgbmv

Syntax

```
#include "armpl.h"

void cblas_cgbmv(const enum CBLAS_ORDER order,
                 const enum CBLAS_TRANSPOSE TransA, const armpl_int_t M,
                 const armpl_int_t N, const armpl_int_t KL,
                 const armpl_int_t KU, const armpl_singlecomplex_t*alpha,
                 const armpl_singlecomplex_t*A, const armpl_int_t lda,
                 const armpl_singlecomplex_t*X, const armpl_int_t incX,
                 const armpl_singlecomplex_t*beta, armpl_singlecomplex_t*Y,
                 const armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dgbmv*, *cblas_sgbmv* and *cblas_zgbmv*. It also exists with a native Fortran interface as *cgbmv*.

3.5.5 cblas_cgemm

Syntax

```
#include "armpl.h"

void cblas_cgemm(const enum CBLAS_ORDER Order,
                 const enum CBLAS_TRANSPOSE TransA,
                 const enum CBLAS_TRANSPOSE TransB, const armpl_int_t M,
                 const armpl_int_t N, const armpl_int_t K,
                 const armpl_singlecomplex_t*alpha,
                 const armpl_singlecomplex_t*A, const armpl_int_t lda,
                 const armpl_singlecomplex_t*B, const armpl_int_t ldb,
                 const armpl_singlecomplex_t*beta, armpl_singlecomplex_t*C,
                 const armpl_int_t ldc);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dgemm*, *cblas_sgemm* and *cblas_zgemm*. It also exists with a native Fortran interface as *cgemm*.

3.5.6 cblas_cgemm3m

Syntax

```
#include "armpl.h"

void cblas_cgemm3m(const enum CBLAS_ORDER Order,
                   const enum CBLAS_TRANSPOSE TransA,
                   const enum CBLAS_TRANSPOSE TransB, const armpl_int_t M,
```

(continues on next page)

(continued from previous page)

```
                    const armpl_int_t N, const armpl_int_t K,
                    const armpl_singlecomplex_t *alpha,
                    const armpl_singlecomplex_t *A, const armpl_int_t lda,
                    const armpl_singlecomplex_t *B, const armpl_int_t ldb,
                    const armpl_singlecomplex_t *beta,
                    armpl_singlecomplex_t *C, const armpl_int_t ldc);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_zgemm3m*. It also exists with a native Fortran interface as *cgemm3m*.

3.5.7 cblas_cgemv

Syntax

```
#include "armpl.h"

void cblas_cgemv(const enum CBLAS_ORDER order,
                 const enum CBLAS_TRANSPOSE TransA, const armpl_int_t M,
                 const armpl_int_t N, const armpl_singlecomplex_t*alpha,
                 const armpl_singlecomplex_t*A, const armpl_int_t lda,
                 const armpl_singlecomplex_t*X, const armpl_int_t incX,
                 const armpl_singlecomplex_t*beta, armpl_singlecomplex_t*Y,
                 const armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dgemv*, *cblas_sgemv* and *cblas_zgemv*. It also exists with a native Fortran interface as *cgemv*.

3.5.8 cblas_cgerc

Syntax

```
#include "armpl.h"

void cblas_cgerc(const enum CBLAS_ORDER order, const armpl_int_t M,
                 const armpl_int_t N, const armpl_singlecomplex_t*alpha,
                 const armpl_singlecomplex_t*X, const armpl_int_t incX,
                 const armpl_singlecomplex_t*Y, const armpl_int_t incY,
                 armpl_singlecomplex_t*A, const armpl_int_t lda);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_zgerc*. It also exists with a native Fortran interface as *cgerc*.

3.5.9 cblas_cgeru

Syntax

```
#include "armpl.h"

void cblas_cgeru(const enum CBLAS_ORDER order, const armpl_int_t M,
                 const armpl_int_t N, const armpl_singlecomplex_t*alpha,
                 const armpl_singlecomplex_t*X, const armpl_int_t incX,
                 const armpl_singlecomplex_t*Y, const armpl_int_t incY,
                 armpl_singlecomplex_t*A, const armpl_int_t lda);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_zgeru*. It also exists with a native Fortran interface as *cgeru*.

3.5.10 cblas_chbmv

Syntax

```
#include "armpl.h"

void cblas_chbmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const armpl_int_t N, const armpl_int_t K,
                 const armpl_singlecomplex_t*alpha,
                 const armpl_singlecomplex_t*A, const armpl_int_t lda,
                 const armpl_singlecomplex_t*X, const armpl_int_t incX,
                 const armpl_singlecomplex_t*beta, armpl_singlecomplex_t*Y,
                 const armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_zhbmv*. It also exists with a native Fortran interface as *chbmv*.

3.5.11 cblas_chemm

Syntax

```
#include "armpl.h"

void cblas_chemm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side,
                 const enum CBLAS_UPLO Uplo, const armpl_int_t M,
                 const armpl_int_t N, const armpl_singlecomplex_t*alpha,
                 const armpl_singlecomplex_t*A, const armpl_int_t lda,
```

(continues on next page)

(continued from previous page)

```
                const armpl_singlecomplex_t*B, const armpl_int_t ldb,
                const armpl_singlecomplex_t*beta, armpl_singlecomplex_t*C,
                const armpl_int_t ldc);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_zhemm*. It also exists with a native Fortran interface as *chemm*.

3.5.12 cblas_chemv

Syntax

```
#include "armpl.h"

void cblas_chemv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                const armpl_int_t N, const armpl_singlecomplex_t*alpha,
                const armpl_singlecomplex_t*A, const armpl_int_t lda,
                const armpl_singlecomplex_t*X, const armpl_int_t incX,
                const armpl_singlecomplex_t*beta, armpl_singlecomplex_t*Y,
                const armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_zhemv*. It also exists with a native Fortran interface as *chemv*.

3.5.13 cblas_cher

Syntax

```
#include "armpl.h"

void cblas_cher(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                const armpl_int_t N, const float alpha,
                const armpl_singlecomplex_t*X, const armpl_int_t incX,
                armpl_singlecomplex_t*A, const armpl_int_t lda);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_zher*. It also exists with a native Fortran interface as *cher*.

3.5.14 cblas_cher2

Syntax

```
#include "armpl.h"

void cblas_cher2(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const armpl_int_t N, const armpl_singlecomplex_t*alpha,
                 const armpl_singlecomplex_t*X, const armpl_int_t incX,
                 const armpl_singlecomplex_t*Y, const armpl_int_t incY,
                 armpl_singlecomplex_t*A, const armpl_int_t lda);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_zher2*. It also exists with a native Fortran interface as *cher2*.

3.5.15 cblas_cher2k

Syntax

```
#include "armpl.h"

void cblas_cher2k(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo,
                  const enum CBLAS_TRANSPOSE Trans, const armpl_int_t N,
                  const armpl_int_t K, const armpl_singlecomplex_t*alpha,
                  const armpl_singlecomplex_t*A, const armpl_int_t lda,
                  const armpl_singlecomplex_t*B, const armpl_int_t ldb,
                  const float beta, armpl_singlecomplex_t*C,
                  const armpl_int_t ldc);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_zher2k*. It also exists with a native Fortran interface as *cher2k*.

3.5.16 cblas_cherk

Syntax

```
#include "armpl.h"

void cblas_cherk(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE Trans, const armpl_int_t N,
                 const armpl_int_t K, const float alpha,
                 const armpl_singlecomplex_t*A, const armpl_int_t lda,
                 const float beta, armpl_singlecomplex_t*C,
                 const armpl_int_t ldc);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_zherk*. It also exists with a native Fortran interface as *cherk*.

3.5.17 cblas_chpmv

Syntax

```
#include "armpl.h"

void cblas_chpmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const armpl_int_t N, const armpl_singlecomplex_t*alpha,
                 const armpl_singlecomplex_t*Ap,
                 const armpl_singlecomplex_t*X, const armpl_int_t incX,
                 const armpl_singlecomplex_t*beta, armpl_singlecomplex_t*Y,
                 const armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_zhpmv*. It also exists with a native Fortran interface as *chpmv*.

3.5.18 cblas_chpr

Syntax

```
#include "armpl.h"

void cblas_chpr(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                const armpl_int_t N, const float alpha,
                const armpl_singlecomplex_t*X, const armpl_int_t incX,
                armpl_singlecomplex_t*A);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_zhpr*. It also exists with a native Fortran interface as *chpr*.

3.5.19 cblas_chpr2

Syntax

```
#include "armpl.h"

void cblas_chpr2(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const armpl_int_t N, const armpl_singlecomplex_t*alpha,
                 const armpl_singlecomplex_t*X, const armpl_int_t incX,
                 const armpl_singlecomplex_t*Y, const armpl_int_t incY,
                 armpl_singlecomplex_t*Ap);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_zhpr2*. It also exists with a native Fortran interface as *chpr2*.

3.5.20 cblas_cscal

Syntax

```
#include "armpl.h"

void cblas_cscal(const armpl_int_t N, const armpl_singlecomplex_t*alpha,
                 armpl_singlecomplex_t*X, const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dscal*, *cblas_sscal* and *cblas_zscal*. It also exists with a native Fortran interface as *cscal*.

3.5.21 cblas_csscal

Syntax

```
#include "armpl.h"

void cblas_csscal(const armpl_int_t N, const float alpha,
                  armpl_singlecomplex_t*X, const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

It also exists with a native Fortran interface as *csscal*.

3.5.22 cblas_cswap

Syntax

```
#include "armpl.h"

void cblas_cswap(const armpl_int_t N, armpl_singlecomplex_t*X,
                 const armpl_int_t incX, armpl_singlecomplex_t*Y,
                 const armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dswap*, *cblas_sswap* and *cblas_zswap*. It also exists with a native Fortran interface as *cswap*.

3.5.23 cblas_csymm

Syntax

```
#include "armpl.h"

void cblas_csymm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side,
                 const enum CBLAS_UPLO Uplo, const armpl_int_t M,
                 const armpl_int_t N, const armpl_singlecomplex_t*alpha,
                 const armpl_singlecomplex_t*A, const armpl_int_t lda,
                 const armpl_singlecomplex_t*B, const armpl_int_t ldb,
                 const armpl_singlecomplex_t*beta, armpl_singlecomplex_t*C,
                 const armpl_int_t ldc);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dsymm*, *cblas_ssymm* and *cblas_zsymm*. It also exists with a native Fortran interface as *csymm*.

3.5.24 cblas_csyr2k

Syntax

```
#include "armpl.h"

void cblas_csyr2k(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo,
                  const enum CBLAS_TRANSPOSE Trans, const armpl_int_t N,
                  const armpl_int_t K, const armpl_singlecomplex_t*alpha,
                  const armpl_singlecomplex_t*A, const armpl_int_t lda,
                  const armpl_singlecomplex_t*B, const armpl_int_t ldb,
                  const armpl_singlecomplex_t*beta, armpl_singlecomplex_t*C,
                  const armpl_int_t ldc);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dsyr2k*, *cblas_ssyr2k* and *cblas_zsyr2k*. It also exists with a native Fortran interface as *csyr2k*.

3.5.25 cblas_csyrk

Syntax

```
#include "armpl.h"

void cblas_csyrk(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE Trans, const armpl_int_t N,
                 const armpl_int_t K, const armpl_singlecomplex_t*alpha,
                 const armpl_singlecomplex_t*A, const armpl_int_t lda,
                 const armpl_singlecomplex_t*beta, armpl_singlecomplex_t*C,
                 const armpl_int_t ldc);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dsyrk*, *cblas_ssyrk* and *cblas_zsyrk*. It also exists with a native Fortran interface as *csyrk*.

3.5.26 cblas_ctbmv

Syntax

```
#include "armpl.h"

void cblas_ctbmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE TransA,
                 const enum CBLAS_DIAG Diag, const armpl_int_t N,
                 const armpl_int_t K, const armpl_singlecomplex_t*A,
                 const armpl_int_t lda, armpl_singlecomplex_t*X,
                 const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dtbmv*, *cblas_stbmv* and *cblas_ztbmv*. It also exists with a native Fortran interface as *ctbmv*.

3.5.27 cblas_ctbsv

Syntax

```
#include "armpl.h"

void cblas_ctbsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE TransA,
                 const enum CBLAS_DIAG Diag, const armpl_int_t N,
                 const armpl_int_t K, const armpl_singlecomplex_t*A,
                 const armpl_int_t lda, armpl_singlecomplex_t*X,
                 const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dtbsv*, *cblas_stbsv* and *cblas_ztbsv*. It also exists with a native Fortran interface as *ctbsv*.

3.5.28 cblas_ctpmv

Syntax

```
#include "armpl.h"

void cblas_ctpmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE TransA,
                 const enum CBLAS_DIAG Diag, const armpl_int_t N,
                 const armpl_singlecomplex_t*Ap, armpl_singlecomplex_t*X,
                 const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dtpmv*, *cblas_stpmv* and *cblas_ztpmv*. It also exists with a native Fortran interface as *ctpmv*.

3.5.29 cblas_ctpsv

Syntax

```
#include "armpl.h"

void cblas_ctpsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE TransA,
                 const enum CBLAS_DIAG Diag, const armpl_int_t N,
                 const armpl_singlecomplex_t*Ap, armpl_singlecomplex_t*X,
                 const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dtpsv*, *cblas_stpsv* and *cblas_ztpsv*. It also exists with a native Fortran interface as *ctpsv*.

3.5.30 cblas_ctrmm

Syntax

```
#include "armpl.h"

void cblas_ctrmm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side,
                 const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE TransA,
                 const enum CBLAS_DIAG Diag, const armpl_int_t M,
                 const armpl_int_t N, const armpl_singlecomplex_t*alpha,
                 const armpl_singlecomplex_t*A, const armpl_int_t lda,
                 armpl_singlecomplex_t*B, const armpl_int_t ldb);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dtrmm*, *cblas_strmm* and *cblas_ztrmm*. It also exists with a native Fortran interface as *ctrmm*.

3.5.31 cblas_ctrmv

Syntax

```
#include "armpl.h"

void cblas_ctrmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE TransA,
                 const enum CBLAS_DIAG Diag, const armpl_int_t N,
                 const armpl_singlecomplex_t*A, const armpl_int_t lda,
                 armpl_singlecomplex_t*X, const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dtrmv*, *cblas_strmv* and *cblas_ztrmv*. It also exists with a native Fortran interface as *ctrmv*.

3.5.32 cblas_ctrsm

Syntax

```
#include "armpl.h"

void cblas_ctrsm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side,
                 const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE TransA,
                 const enum CBLAS_DIAG Diag, const armpl_int_t M,
                 const armpl_int_t N, const armpl_singlecomplex_t*alpha,
                 const armpl_singlecomplex_t*A, const armpl_int_t lda,
                 armpl_singlecomplex_t*B, const armpl_int_t ldb);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dtrsm*, *cblas_strsm* and *cblas_ztrsm*. It also exists with a native Fortran interface as *ctrsm*.

3.5.33 cblas_ctrsv

Syntax

```
#include "armpl.h"

void cblas_ctrsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE TransA,
                 const enum CBLAS_DIAG Diag, const armpl_int_t N,
                 const armpl_singlecomplex_t*A, const armpl_int_t lda,
                 armpl_singlecomplex_t*X, const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dtrsv*, *cblas_strsv* and *cblas_ztrsv*. It also exists with a native Fortran interface as *ctrsv*.

3.5.34 cblas_cwaxpby

Syntax

```
#include "armpl.h"

void cblas_cwaxpby(armpl_int_t N, const armpl_singlecomplex_t *alpha,
                   const armpl_singlecomplex_t *X, armpl_int_t incX,
                   const armpl_singlecomplex_t *beta,
                   armpl_singlecomplex_t *Y, armpl_int_t incY,
                   armpl_singlecomplex_t *W, armpl_int_t incw);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dwaxpby*, *cblas_swaxpby* and *cblas_zwaxpby*. It also exists with a native Fortran interface as *cwaxpby*.

3.5.35 cblas_dasum

Syntax

```
#include "armpl.h"

double cblas_dasum(const armpl_int_t N, const double *X,
                   const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dasum* and *cblas_sasum*. It also exists with a native Fortran interface as *dasum*.

3.5.36 cblas_caxpby

Syntax

```
#include "armpl.h"

void cblas_daxpby(armpl_int_t N, const double *alpha,const double *X,
                  armpl_int_t incX, const double *beta, double *Y,
                  armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_caxpby*, *cblas_saxpby* and *cblas_zaxpby*. It also exists with a native Fortran interface as *daxpby*.

3.5.37 cblas_daxpy

Syntax

```
#include "armpl.h"

void cblas_daxpy(const armpl_int_t N, const double alpha, const double *X,
                 const armpl_int_t incX, double *Y, const armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_daxpy*, *cblas_saxpy* and *cblas_zaxpy*. It also exists with a native Fortran interface as *daxpy*.

3.5.38 cblas_dcopy

Syntax

```
#include "armpl.h"

void cblas_dcopy(const armpl_int_t N, const double *X, const armpl_int_t incX,
                 double *Y, const armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dcopy*, *cblas_scopy* and *cblas_zcopy*. It also exists with a native Fortran interface as *dcopy*.

3.5.39 cblas_ddot

Syntax

```
#include "armpl.h"

double cblas_ddot(const armpl_int_t N, const double *X,
                  const armpl_int_t incX, const double *Y,
                  const armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_ddot* and *cblas_sdot*. It also exists with a native Fortran interface as *ddot*.

3.5.40 cblas_dgbmv

Syntax

```
#include "armpl.h"

void cblas_dgbmv(const enum CBLAS_ORDER order,
                 const enum CBLAS_TRANSPOSE TransA, const armpl_int_t M,
                 const armpl_int_t N, const armpl_int_t KL,
                 const armpl_int_t KU, const double alpha, const double *A,
                 const armpl_int_t lda, const double *X,
                 const armpl_int_t incX, const double beta, double *Y,
                 const armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dgbmv*, *cblas_sgbmv* and *cblas_zgbmv*. It also exists with a native Fortran interface as *dgbmv*.

3.5.41 cblas_dgemm

Syntax

```
#include "armpl.h"

void cblas_dgemm(const enum CBLAS_ORDER Order,
                 const enum CBLAS_TRANSPOSE TransA,
                 const enum CBLAS_TRANSPOSE TransB, const armpl_int_t M,
                 const armpl_int_t N, const armpl_int_t K, const double alpha,
                 const double *A, const armpl_int_t lda, const double *B,
                 const armpl_int_t ldb, const double beta, double *C,
                 const armpl_int_t ldc);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dgemm*, *cblas_sgemm* and *cblas_zgemm*. It also exists with a native Fortran interface as *dgemm*.

3.5.42 cblas_dgemv

Syntax

```
#include "armpl.h"

void cblas_dgemv(const enum CBLAS_ORDER order,
                 const enum CBLAS_TRANSPOSE TransA, const armpl_int_t M,
                 const armpl_int_t N, const double alpha, const double *A,
                 const armpl_int_t lda, const double *X,
                 const armpl_int_t incX, const double beta, double *Y,
                 const armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dgemv*, *cblas_sgemv* and *cblas_zgemv*. It also exists with a native Fortran interface as *dgemv*.

3.5.43 cblas_dger

Syntax

```
#include "armpl.h"

void cblas_dger(const enum CBLAS_ORDER order, const armpl_int_t M,
                const armpl_int_t N, const double alpha, const double *X,
                const armpl_int_t incX, const double *Y,
                const armpl_int_t incY, double *A, const armpl_int_t lda);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dger* and *cblas_sger*. It also exists with a native Fortran interface as *dger*.

3.5.44 cblas_dnrm2

Syntax

```
#include "armpl.h"

double cblas_dnrm2(const armpl_int_t N, const double *X,
                   const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dnrm2* and *cblas_snrm2*. It also exists with a native Fortran interface as *dnrm2*.

3.5.45 cblas_drot

Syntax

```
#include "armpl.h"

void cblas_drot(const armpl_int_t N, double *X, const armpl_int_t incX,
                double *Y, const armpl_int_t incY, const double c,
                const double s);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_drot* and *cblas_srot*. It also exists with a native Fortran interface as *drot*.

3.5.46 cblas_drotg

Syntax

```
#include "armpl.h"

void cblas_drotg(double *a, double *b, double *c, double *s);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_drotg* and *cblas_srotg*. It also exists with a native Fortran interface as *drotg*.

3.5.47 cblas_drotm

Syntax

```
#include "armpl.h"

void cblas_drotm(const armpl_int_t N, double *X, const armpl_int_t incX,
                 double *Y, const armpl_int_t incY, const double *P);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_drotm* and *cblas_srotm*. It also exists with a native Fortran interface as *drotm*.

3.5.48 cblas_drotmg

Syntax

```
#include "armpl.h"

void cblas_drotmg(double *d1, double *d2, double *b1, const double b2,
                  double *P);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_drotmg* and *cblas_srotmg*. It also exists with a native Fortran interface as *drotmg*.

3.5.49 cblas_dsbmv

Syntax

```
#include "armpl.h"

void cblas_dsbmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const armpl_int_t N, const armpl_int_t K, const double alpha,
                 const double *A, const armpl_int_t lda, const double *X,
                 const armpl_int_t incX, const double beta, double *Y,
                 const armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dsbmv* and *cblas_ssbmv*. It also exists with a native Fortran interface as *dsbmv*.

3.5.50 cblas_dscal

Syntax

```
#include "armpl.h"

void cblas_dscal(const armpl_int_t N, const double alpha, double *X,
                 const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dscal*, *cblas_sscal* and *cblas_zscal*. It also exists with a native Fortran interface as *dscal*.

3.5.51 cblas_dsdot

Syntax

```
#include "armpl.h"

double cblas_dsdot(const armpl_int_t N, const float *X,
                   const armpl_int_t incX, const float *Y,
                   const armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dsdot*. It also exists with a native Fortran interface as *dsdot*.

3.5.52 cblas_dspmv

Syntax

```
#include "armpl.h"

void cblas_dspmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const armpl_int_t N, const double alpha, const double *Ap,
                 const double *X, const armpl_int_t incX, const double beta,
                 double *Y, const armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dspmv* and *cblas_sspmv*. It also exists with a native Fortran interface as *dspmv*.

3.5.53 cblas_dspr

Syntax

```
#include "armpl.h"

void cblas_dspr(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                const armpl_int_t N, const double alpha, const double *X,
                const armpl_int_t incX, double *Ap);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dspr* and *cblas_sspr*. It also exists with a native Fortran interface as *dspr*.

3.5.54 cblas_dspr2

Syntax

```
#include "armpl.h"

void cblas_dspr2(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const armpl_int_t N, const double alpha, const double *X,
                 const armpl_int_t incX, const double *Y,
                 const armpl_int_t incY, double *A);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dspr2* and *cblas_sspr2*. It also exists with a native Fortran interface as *dspr2*.

3.5.55 cblas_dswap

Syntax

```
#include "armpl.h"

void cblas_dswap(const armpl_int_t N, double *X, const armpl_int_t incX,
                 double *Y, const armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dswap*, *cblas_sswap* and *cblas_zswap*. It also exists with a native Fortran interface as *dswap*.

3.5.56 cblas_dsymm

Syntax

```
#include "armpl.h"

void cblas_dsymm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side,
                 const enum CBLAS_UPLO Uplo, const armpl_int_t M,
                 const armpl_int_t N, const double alpha, const double *A,
                 const armpl_int_t lda, const double *B,
                 const armpl_int_t ldb, const double beta, double *C,
                 const armpl_int_t ldc);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dsymm*, *cblas_ssymm* and *cblas_zsymm*. It also exists with a native Fortran interface as *dsymm*.

3.5.57 cblas_dsymv

Syntax

```
#include "armpl.h"

void cblas_dsymv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const armpl_int_t N, const double alpha, const double *A,
                 const armpl_int_t lda, const double *X,
                 const armpl_int_t incX, const double beta, double *Y,
                 const armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dsymv* and *cblas_ssymv*. It also exists with a native Fortran interface as *dsymv*.

3.5.58 cblas_dsyr

Syntax

```
#include "armpl.h"

void cblas_dsyr(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                const armpl_int_t N, const double alpha, const double *X,
                const armpl_int_t incX, double *A, const armpl_int_t lda);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dsyr* and *cblas_ssyr*. It also exists with a native Fortran interface as *dsyr*.

3.5.59 cblas_dsyr2

Syntax

```
#include "armpl.h"

void cblas_dsyr2(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const armpl_int_t N, const double alpha, const double *X,
                 const armpl_int_t incX, const double *Y,
                 const armpl_int_t incY, double *A, const armpl_int_t lda);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dsyr2* and *cblas_ssyr2*. It also exists with a native Fortran interface as *dsyr2*.

3.5.60 cblas_dsyr2k

Syntax

```
#include "armpl.h"

void cblas_dsyr2k(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo,
                  const enum CBLAS_TRANSPOSE Trans, const armpl_int_t N,
                  const armpl_int_t K, const double alpha, const double *A,
                  const armpl_int_t lda, const double *B,
                  const armpl_int_t ldb, const double beta, double *C,
                  const armpl_int_t ldc);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dsyr2k*, *cblas_ssyr2k* and *cblas_zsyr2k*. It also exists with a native Fortran interface as *dsyr2k*.

3.5.61 cblas_dsyrk

Syntax

```
#include "armpl.h"

void cblas_dsyrk(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE Trans, const armpl_int_t N,
                 const armpl_int_t K, const double alpha, const double *A,
                 const armpl_int_t lda, const double beta, double *C,
                 const armpl_int_t ldc);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dsyrk*, *cblas_ssyrk* and *cblas_zsyrk*. It also exists with a native Fortran interface as *dsyrk*.

3.5.62 cblas_dtbmv

Syntax

```
#include "armpl.h"

void cblas_dtbmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE TransA,
                 const enum CBLAS_DIAG Diag, const armpl_int_t N,
                 const armpl_int_t K, const double *A, const armpl_int_t lda,
                 double *X, const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dtbmv*, *cblas_stbmv* and *cblas_ztbmv*. It also exists with a native Fortran interface as *dtbmv*.

3.5.63 cblas_dtbsv

Syntax

```
#include "armpl.h"

void cblas_dtbsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE TransA,
                 const enum CBLAS_DIAG Diag, const armpl_int_t N,
                 const armpl_int_t K, const double *A, const armpl_int_t lda,
                 double *X, const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dtbsv*, *cblas_stbsv* and *cblas_ztbsv*. It also exists with a native Fortran interface as *dtbsv*.

3.5.64 cblas_dtpmv

Syntax

```
#include "armpl.h"

void cblas_dtpmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE TransA,
                 const enum CBLAS_DIAG Diag, const armpl_int_t N,
                 const double *Ap, double *X, const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dtpmv*, *cblas_stpmv* and *cblas_ztpmv*. It also exists with a native Fortran interface as *dtpmv*.

3.5.65 cblas_dtpsv

Syntax

```
#include "armpl.h"

void cblas_dtpsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE TransA,
                 const enum CBLAS_DIAG Diag, const armpl_int_t N,
                 const double *Ap, double *X, const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

3.5.66 cblas_dtrmm

Syntax

```
#include "armpl.h"

void cblas_dtrmm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side,
                 const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE TransA,
                 const enum CBLAS_DIAG Diag, const armpl_int_t M,
                 const armpl_int_t N, const double alpha, const double *A,
                 const armpl_int_t lda, double *B, const armpl_int_t ldb);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dtrmm*, *cblas_strmm* and *cblas_ztrmm*. It also exists with a native Fortran interface as *dtrmm*.

3.5.67 cblas_dtrmv

Syntax

```
#include "armpl.h"

void cblas_dtrmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE TransA,
                 const enum CBLAS_DIAG Diag, const armpl_int_t N,
                 const double *A, const armpl_int_t lda, double *X,
                 const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dtrmv*, *cblas_strmv* and *cblas_ztrmv*. It also exists with a native Fortran interface as *dtrmv*.

3.5.68 cblas_dtrsm

Syntax

```
#include "armpl.h"

void cblas_dtrsm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side,
                 const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE TransA,
                 const enum CBLAS_DIAG Diag, const armpl_int_t M,
                 const armpl_int_t N, const double alpha, const double *A,
                 const armpl_int_t lda, double *B, const armpl_int_t ldb);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dtrsm*, *cblas_strsm* and *cblas_ztrsm*. It also exists with a native Fortran interface as *dtrsm*.

3.5.69 cblas_dtrsv

Syntax

```
#include "armpl.h"

void cblas_dtrsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE TransA,
                 const enum CBLAS_DIAG Diag, const armpl_int_t N,
                 const double *A, const armpl_int_t lda, double *X,
                 const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dtrsv*, *cblas_strsv* and *cblas_ztrsv*. It also exists with a native Fortran interface as *dtrsv*.

3.5.70 cblas_dwaxpby

Syntax

```
#include "armpl.h"

void cblas_dwaxpby(armpl_int_t N, const double *alpha,
                   const double *X, armpl_int_t incX,
```

(continues on next page)

(continued from previous page)

```
                    const double *beta,
                    double *Y, armpl_int_t incY,
                    double *W, armpl_int_t incw);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_cwaxpby*, *cblas_swaxpby* and *cblas_zwaxpby*. It also exists with a native Fortran interface as *dwaxpby*.

3.5.71 cblas_dzasum

Syntax

```
#include "armpl.h"

double cblas_dzasum(const armpl_int_t N, const armpl_doublecomplex_t *X,
                    const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dzasum*. It also exists with a native Fortran interface as *dzasum*.

3.5.72 cblas_dznrm2

Syntax

```
#include "armpl.h"

double cblas_dznrm2(const armpl_int_t N, const armpl_doublecomplex_t *X,
                    const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dznrm2*. It also exists with a native Fortran interface as *dznrm2*.

3.5.73 cblas_icamax

Syntax

```
#include "armpl.h"

armpl_int_t cblas_icamax(const armpl_int_t N, const armpl_singlecomplex_t*X,
                         const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

It also exists with a native Fortran interface as *icamax*.

3.5.74 cblas_idamax

Syntax

```
#include "armpl.h"

armpl_int_t cblas_idamax(const armpl_int_t N, const double *X,
                         const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

It also exists with a native Fortran interface as *idamax*.

3.5.75 cblas_isamax

Syntax

```
#include "armpl.h"

armpl_int_t cblas_isamax(const armpl_int_t N, const float *X,
                         const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

It also exists with a native Fortran interface as *isamax*.

3.5.76 cblas_izamax

Syntax

```
#include "armpl.h"

armpl_int_t cblas_izamax(const armpl_int_t N, const armpl_doublecomplex_t *X,
                         const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

It also exists with a native Fortran interface as *izamax*.

3.5.77 cblas_sasum

Syntax

```
#include "armpl.h"

float cblas_sasum(const armpl_int_t N, const float *X,
                  const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dasum* and *cblas_sasum*. It also exists with a native Fortran interface as *sasum*.

3.5.78 cblas_saxpby

Syntax

```
#include "armpl.h"

void cblas_saxpby(armpl_int_t N, const float *alpha,const float *X,
                  armpl_int_t incX, const float *beta, float *Y,
                  armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_caxpby*, *cblas_caxpby* and *cblas_zaxpby*. It also exists with a native Fortran interface as *saxpby*.

3.5.79 cblas_saxpy

Syntax

```
#include "armpl.h"

void cblas_saxpy(const armpl_int_t N, const float alpha, const float *X,
                 const armpl_int_t incX, float *Y, const armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_daxpy*, *cblas_saxpy* and *cblas_zaxpy*. It also exists with a native Fortran interface as *saxpy*.

3.5.80 cblas_scasum

Syntax

```
#include "armpl.h"

float cblas_scasum(const armpl_int_t N, const armpl_singlecomplex_t*X,
                   const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_scasum*. It also exists with a native Fortran interface as *scasum*.

3.5.81 cblas_scnrm2

Syntax

```
#include "armpl.h"

float cblas_scnrm2(const armpl_int_t N, const armpl_singlecomplex_t*X,
                   const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_scnrm2*. It also exists with a native Fortran interface as *scnrm2*.

3.5.82 cblas_scopy

Syntax

```
#include "armpl.h"

void cblas_scopy(const armpl_int_t N, const float *X, const armpl_int_t incX,
                 float *Y, const armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dcopy*, *cblas_scopy* and *cblas_zcopy*. It also exists with a native Fortran interface as *scopy*.

3.5.83 cblas_sdot

Syntax

```
#include "armpl.h"

float cblas_sdot(const armpl_int_t N, const float *X, const armpl_int_t incX,
                 const float *Y, const armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_ddot* and *cblas_sdot*. It also exists with a native Fortran interface as *sdot*.

3.5.84 cblas_sdsdot

Syntax

```
#include "armpl.h"

float cblas_sdsdot(const armpl_int_t N, const float alpha, const float *X,
                   const armpl_int_t incX, const float *Y,
                   const armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_sdsdot*. It also exists with a native Fortran interface as *sdsdot*.

3.5.85 cblas_sgbmv

Syntax

```
#include "armpl.h"

void cblas_sgbmv(const enum CBLAS_ORDER order,
                 const enum CBLAS_TRANSPOSE TransA, const armpl_int_t M,
                 const armpl_int_t N, const armpl_int_t KL,
                 const armpl_int_t KU, const float alpha, const float *A,
                 const armpl_int_t lda, const float *X,
                 const armpl_int_t incX, const float beta, float *Y,
                 const armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dgbmv*, *cblas_sgbmv* and *cblas_zgbmv*. It also exists with a native Fortran interface as *sgbmv*.

3.5.86 cblas_sgemm

Syntax

```
#include "armpl.h"

void cblas_sgemm(const enum CBLAS_ORDER Order,
                 const enum CBLAS_TRANSPOSE TransA,
                 const enum CBLAS_TRANSPOSE TransB, const armpl_int_t M,
                 const armpl_int_t N, const armpl_int_t K, const float alpha,
                 const float *A, const armpl_int_t lda, const float *B,
                 const armpl_int_t ldb, const float beta, float *C,
                 const armpl_int_t ldc);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dgemm*, *cblas_sgemm* and *cblas_zgemm*. It also exists with a native Fortran interface as *sgemm*.

3.5.87 cblas_sgemv

Syntax

```
#include "armpl.h"

void cblas_sgemv(const enum CBLAS_ORDER order,
                 const enum CBLAS_TRANSPOSE TransA, const armpl_int_t M,
                 const armpl_int_t N, const float alpha, const float *A,
                 const armpl_int_t lda, const float *X,
                 const armpl_int_t incX, const float beta, float *Y,
                 const armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dgemv*, *cblas_sgemv* and *cblas_zgemv*. It also exists with a native Fortran interface as *sgemv*.

3.5.88 cblas_sger

Syntax

```
#include "armpl.h"

void cblas_sger(const enum CBLAS_ORDER order, const armpl_int_t M,
                const armpl_int_t N, const float alpha, const float *X,
                const armpl_int_t incX, const float *Y,
                const armpl_int_t incY, float *A, const armpl_int_t lda);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dger* and *cblas_sger*. It also exists with a native Fortran interface as *sger*.

3.5.89 cblas_snrm2

Syntax

```
#include "armpl.h"

float cblas_snrm2(const armpl_int_t N, const float *X,
                  const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dnrm2* and *cblas_snrm2*. It also exists with a native Fortran interface as *snrm2*.

3.5.90 cblas_srot

Syntax

```
#include "armpl.h"

void cblas_srot(const armpl_int_t N, float *X, const armpl_int_t incX,
                float *Y, const armpl_int_t incY, const float c,
                const float s);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_drot* and *cblas_srot*. It also exists with a native Fortran interface as *srot*.

3.5.91 cblas_srotg

Syntax

```
#include "armpl.h"

void cblas_srotg(float *a, float *b, float *c, float *s);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_drotg* and *cblas_srotg*. It also exists with a native Fortran interface as *srotg*.

3.5.92 cblas_srotm

Syntax

```
#include "armpl.h"

void cblas_srotm(const armpl_int_t N, float *X, const armpl_int_t incX,
                 float *Y, const armpl_int_t incY, const float *P);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_drotm* and *cblas_srotm*. It also exists with a native Fortran interface as *srotm*.

3.5.93 cblas_srotmg

Syntax

```
#include "armpl.h"

void cblas_srotmg(float *d1, float *d2, float *b1, const float b2, float *P);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_drotmg* and *cblas_srotmg*. It also exists with a native Fortran interface as *srotmg*.

3.5.94 cblas_ssbmv

Syntax

```
#include "armpl.h"

void cblas_ssbmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const armpl_int_t N, const armpl_int_t K, const float alpha,
                 const float *A, const armpl_int_t lda, const float *X,
                 const armpl_int_t incX, const float beta, float *Y,
                 const armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dsbmv* and *cblas_ssbmv*. It also exists with a native Fortran interface as *ssbmv*.

3.5.95 cblas_sscal

Syntax

```
#include "armpl.h"

void cblas_sscal(const armpl_int_t N, const float alpha, float *X,
                 const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dscal*, *cblas_sscal* and *cblas_zscal*. It also exists with a native Fortran interface as *sscal*.

3.5.96 cblas_sspmv

Syntax

```
#include "armpl.h"

void cblas_sspmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const armpl_int_t N, const float alpha, const float *Ap,
                 const float *X, const armpl_int_t incX, const float beta,
                 float *Y, const armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dspmv* and *cblas_sspmv*. It also exists with a native Fortran interface as *sspmv*.

3.5.97 cblas_sspr

Syntax

```
#include "armpl.h"

void cblas_sspr(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                const armpl_int_t N, const float alpha, const float *X,
                const armpl_int_t incX, float *Ap);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dspr* and *cblas_sspr*. It also exists with a native Fortran interface as *sspr*.

3.5.98 cblas_sspr2

Syntax

```
#include "armpl.h"

void cblas_sspr2(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const armpl_int_t N, const float alpha, const float *X,
                 const armpl_int_t incX, const float *Y,
                 const armpl_int_t incY, float *A);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dspr2* and *cblas_sspr2*. It also exists with a native Fortran interface as *sspr2*.

3.5.99 cblas_sswap

Syntax

```
#include "armpl.h"

void cblas_sswap(const armpl_int_t N, float *X, const armpl_int_t incX,
                 float *Y, const armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dswap*, *cblas_sswap* and *cblas_zswap*. It also exists with a native Fortran interface as *sswap*.

3.5.100 cblas_ssymm

Syntax

```
#include "armpl.h"

void cblas_ssymm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side,
                 const enum CBLAS_UPLO Uplo, const armpl_int_t M,
                 const armpl_int_t N, const float alpha, const float *A,
                 const armpl_int_t lda, const float *B, const armpl_int_t ldb,
                 const float beta, float *C, const armpl_int_t ldc);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dsymm*, *cblas_ssymm* and *cblas_zsymm*. It also exists with a native Fortran interface as *ssymm*.

3.5.101 cblas_ssymv

Syntax

```
#include "armpl.h"

void cblas_ssymv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const armpl_int_t N, const float alpha, const float *A,
                 const armpl_int_t lda, const float *X,
                 const armpl_int_t incX, const float beta, float *Y,
                 const armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dsymv* and *cblas_ssymv*. It also exists with a native Fortran interface as *ssymv*.

3.5.102 cblas_ssyr

Syntax

```
#include "armpl.h"

void cblas_ssyr(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                const armpl_int_t N, const float alpha, const float *X,
                const armpl_int_t incX, float *A, const armpl_int_t lda);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dsyr* and *cblas_ssyr*. It also exists with a native Fortran interface as *ssyr*.

3.5.103 cblas_ssyr2

Syntax

```
#include "armpl.h"

void cblas_ssyr2(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const armpl_int_t N, const float alpha, const float *X,
                 const armpl_int_t incX, const float *Y,
                 const armpl_int_t incY, float *A, const armpl_int_t lda);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dsyr2* and *cblas_ssyr2*. It also exists with a native Fortran interface as *ssyr2*.

3.5.104 cblas_ssyr2k

Syntax

```
#include "armpl.h"

void cblas_ssyr2k(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo,
                  const enum CBLAS_TRANSPOSE Trans, const armpl_int_t N,
                  const armpl_int_t K, const float alpha, const float *A,
                  const armpl_int_t lda, const float *B,
                  const armpl_int_t ldb, const float beta, float *C,
                  const armpl_int_t ldc);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dsyr2k*, *cblas_ssyr2k* and *cblas_zsyr2k*. It also exists with a native Fortran interface as *ssyr2k*.

3.5.105 cblas_ssyrk

Syntax

```
#include "armpl.h"

void cblas_ssyrk(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE Trans, const armpl_int_t N,
                 const armpl_int_t K, const float alpha, const float *A,
                 const armpl_int_t lda, const float beta, float *C,
                 const armpl_int_t ldc);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dsyrk*, *cblas_ssyrk* and *cblas_zsyrk*. It also exists with a native Fortran interface as *ssyrk*.

3.5.106 cblas_stbmv

Syntax

```
#include "armpl.h"

void cblas_stbmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE TransA,
                 const enum CBLAS_DIAG Diag, const armpl_int_t N,
                 const armpl_int_t K, const float *A, const armpl_int_t lda,
                 float *X, const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dtbmv*, *cblas_stbmv* and *cblas_ztbmv*. It also exists with a native Fortran interface as *stbmv*.

3.5.107 cblas_stbsv

Syntax

```
#include "armpl.h"

void cblas_stbsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE TransA,
                 const enum CBLAS_DIAG Diag, const armpl_int_t N,
                 const armpl_int_t K, const float *A, const armpl_int_t lda,
                 float *X, const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dtbsv*, *cblas_stbsv* and *cblas_ztbsv*. It also exists with a native Fortran interface as *stbsv*.

3.5.108 cblas_stpmv

Syntax

```
#include "armpl.h"

void cblas_stpmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE TransA,
```

(continues on next page)

(continued from previous page)

```
                const enum CBLAS_DIAG Diag, const armpl_int_t N,
                const float *Ap, float *X, const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dtpmv*, *cblas_stpmv* and *cblas_ztpmv*. It also exists with a native Fortran interface as *stpmv*.

3.5.109 cblas_stpsv

Syntax

```
#include "armpl.h"

void cblas_stpsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE TransA,
                 const enum CBLAS_DIAG Diag, const armpl_int_t N,
                 const float *Ap, float *X, const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dtpsv*, *cblas_stpsv* and *cblas_ztpsv*. It also exists with a native Fortran interface as *stpsv*.

3.5.110 cblas_strmm

Syntax

```
#include "armpl.h"

void cblas_strmm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side,
                 const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE TransA,
                 const enum CBLAS_DIAG Diag, const armpl_int_t M,
                 const armpl_int_t N, const float alpha, const float *A,
                 const armpl_int_t lda, float *B, const armpl_int_t ldb);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dtrmm*, *cblas_strmm* and *cblas_ztrmm*. It also exists with a native Fortran interface as *strmm*.

3.5.111 cblas_strmv

Syntax

```
#include "armpl.h"

void cblas_strmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE TransA,
                 const enum CBLAS_DIAG Diag, const armpl_int_t N,
                 const float *A, const armpl_int_t lda, float *X,
                 const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dtrmv*, *cblas_strmv* and *cblas_ztrmv*. It also exists with a native Fortran interface as *strmv*.

3.5.112 cblas_strsm

Syntax

```
#include "armpl.h"

void cblas_strsm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side,
                 const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE TransA,
                 const enum CBLAS_DIAG Diag, const armpl_int_t M,
                 const armpl_int_t N, const float alpha, const float *A,
                 const armpl_int_t lda, float *B, const armpl_int_t ldb);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dtrsm*, *cblas_strsm* and *cblas_ztrsm*. It also exists with a native Fortran interface as *strsm*.

3.5.113 cblas_strsv

Syntax

```
#include "armpl.h"

void cblas_strsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE TransA,
                 const enum CBLAS_DIAG Diag, const armpl_int_t N,
                 const float *A, const armpl_int_t lda, float *X,
                 const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dtrsv*, *cblas_strsv* and *cblas_ztrsv*. It also exists with a native Fortran interface as *strsv*.

3.5.114 cblas_swaxpby

Syntax

```
#include "armpl.h"

void cblas_swaxpby(armpl_int_t N, const float *alpha,
                   const float *X, armpl_int_t incX,
                   const float *beta,
                   float *Y, armpl_int_t incY,
                   float *W, armpl_int_t incw);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_cwaxpby*, *cblas_dwaxpby* and *cblas_zwaxpby*. It also exists with a native Fortran interface as *swaxpby*.

3.5.115 cblas_xerbla

Syntax

```
#include "armpl.h"

void cblas_xerbla(armpl_int_t RowMajorStrg, armpl_int_t CBLAS_CallFromC,
                  armpl_int_t p, const char *rout, const char *form,
                  int enumval);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

It also exists with a native Fortran interface as *xerbla*.

3.5.116 cblas_zaxpby

Syntax

```
#include "armpl.h"

void cblas_zaxpby(armpl_int_t N, const armpl_doublecomplex_t *alpha,
                  const armpl_doublecomplex_t *X, armpl_int_t incX,
                  const armpl_doublecomplex_t *beta,
                  armpl_doublecomplex_t *Y, armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_caxpby*, *cblas_caxpby* and *cblas_saxpby*. It also exists with a native Fortran interface as *zaxpby*.

3.5.117 cblas_zaxpy

Syntax

```
#include "armpl.h"

void cblas_zaxpy(const armpl_int_t N, const armpl_doublecomplex_t *alpha,
                 const armpl_doublecomplex_t *X, const armpl_int_t incX,
                 armpl_doublecomplex_t *Y, const armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_daxpy*, *cblas_saxpy* and *cblas_zaxpy*. It also exists with a native Fortran interface as *zaxpy*.

3.5.118 cblas_zcopy

Syntax

```
#include "armpl.h"

void cblas_zcopy(const armpl_int_t N, const armpl_doublecomplex_t *X,
                 const armpl_int_t incX, armpl_doublecomplex_t *Y,
                 const armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dcopy*, *cblas_scopy* and *cblas_zcopy*. It also exists with a native Fortran interface as *zcopy*.

3.5.119 cblas_zdscal

Syntax

```
#include "armpl.h"

void cblas_zdscal(const armpl_int_t N, const double alpha,
                  armpl_doublecomplex_t *X, const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_zdscal*. It also exists with a native Fortran interface as *zdscal*.

3.5.120 cblas_zgbmv

Syntax

```
#include "armpl.h"

void cblas_zgbmv(const enum CBLAS_ORDER order,
                 const enum CBLAS_TRANSPOSE TransA, const armpl_int_t M,
                 const armpl_int_t N, const armpl_int_t KL,
                 const armpl_int_t KU, const armpl_doublecomplex_t *alpha,
                 const armpl_doublecomplex_t *A, const armpl_int_t lda,
                 const armpl_doublecomplex_t *X, const armpl_int_t incX,
                 const armpl_doublecomplex_t *beta, armpl_doublecomplex_t *Y,
                 const armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dgbmv*, *cblas_sgbmv* and *cblas_zgbmv*. It also exists with a native Fortran interface as *zgbmv*.

3.5.121 cblas_zgemm

Syntax

```
#include "armpl.h"

void cblas_zgemm(const enum CBLAS_ORDER Order,
                 const enum CBLAS_TRANSPOSE TransA,
                 const enum CBLAS_TRANSPOSE TransB, const armpl_int_t M,
                 const armpl_int_t N, const armpl_int_t K,
                 const armpl_doublecomplex_t *alpha,
                 const armpl_doublecomplex_t *A, const armpl_int_t lda,
                 const armpl_doublecomplex_t *B, const armpl_int_t ldb,
                 const armpl_doublecomplex_t *beta, armpl_doublecomplex_t *C,
                 const armpl_int_t ldc);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dgemm*, *cblas_sgemm* and *cblas_zgemm*. It also exists with a native Fortran interface as *zgemm*.

3.5.122 cblas_zgemm3m

Syntax

```
#include "armpl.h"

void cblas_zgemm3m(const enum CBLAS_ORDER Order,
                   const enum CBLAS_TRANSPOSE TransA,
                   const enum CBLAS_TRANSPOSE TransB, const armpl_int_t M,
                   const armpl_int_t N, const armpl_int_t K,
                   const armpl_doublecomplex_t *alpha,
                   const armpl_doublecomplex_t *A, const armpl_int_t lda,
                   const armpl_doublecomplex_t *B, const armpl_int_t ldb,
                   const armpl_doublecomplex_t *beta,
                   armpl_doublecomplex_t *C, const armpl_int_t ldc);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_cgemm3m*. It also exists with a native Fortran interface as *zgemm3m*.

3.5.123 cblas_zgemv

Syntax

```
#include "armpl.h"

void cblas_zgemv(const enum CBLAS_ORDER order,
                 const enum CBLAS_TRANSPOSE TransA, const armpl_int_t M,
                 const armpl_int_t N, const armpl_doublecomplex_t *alpha,
                 const armpl_doublecomplex_t *A, const armpl_int_t lda,
                 const armpl_doublecomplex_t *X, const armpl_int_t incX,
                 const armpl_doublecomplex_t *beta, armpl_doublecomplex_t *Y,
                 const armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dgemv*, *cblas_sgemv* and *cblas_zgemv*. It also exists with a native Fortran interface as *zgemv*.

3.5.124 cblas_zgerc

Syntax

```
#include "armpl.h"

void cblas_zgerc(const enum CBLAS_ORDER order, const armpl_int_t M,
                 const armpl_int_t N, const armpl_doublecomplex_t *alpha,
                 const armpl_doublecomplex_t *X, const armpl_int_t incX,
                 const armpl_doublecomplex_t *Y, const armpl_int_t incY,
                 armpl_doublecomplex_t *A, const armpl_int_t lda);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_zgerc*. It also exists with a native Fortran interface as *zgerc*.

3.5.125 cblas_zgeru

Syntax

```
#include "armpl.h"

void cblas_zgeru(const enum CBLAS_ORDER order, const armpl_int_t M,
                 const armpl_int_t N, const armpl_doublecomplex_t *alpha,
                 const armpl_doublecomplex_t *X, const armpl_int_t incX,
                 const armpl_doublecomplex_t *Y, const armpl_int_t incY,
                 armpl_doublecomplex_t *A, const armpl_int_t lda);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_zgeru*. It also exists with a native Fortran interface as *zgeru*.

3.5.126 cblas_zhbmv

Syntax

```
#include "armpl.h"

void cblas_zhbmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const armpl_int_t N, const armpl_int_t K,
                 const armpl_doublecomplex_t *alpha,
                 const armpl_doublecomplex_t *A, const armpl_int_t lda,
                 const armpl_doublecomplex_t *X, const armpl_int_t incX,
                 const armpl_doublecomplex_t *beta, armpl_doublecomplex_t *Y,
                 const armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_zhbmv*. It also exists with a native Fortran interface as *zhbmv*.

3.5.127 cblas_zhemm

Syntax

```
#include "armpl.h"

void cblas_zhemm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side,
                 const enum CBLAS_UPLO Uplo, const armpl_int_t M,
                 const armpl_int_t N, const armpl_doublecomplex_t *alpha,
                 const armpl_doublecomplex_t *A, const armpl_int_t lda,
                 const armpl_doublecomplex_t *B, const armpl_int_t ldb,
                 const armpl_doublecomplex_t *beta, armpl_doublecomplex_t *C,
                 const armpl_int_t ldc);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_zhemm*. It also exists with a native Fortran interface as *zhemm*.

3.5.128 cblas_zhemv

Syntax

```
#include "armpl.h"

void cblas_zhemv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const armpl_int_t N, const armpl_doublecomplex_t *alpha,
                 const armpl_doublecomplex_t *A, const armpl_int_t lda,
                 const armpl_doublecomplex_t *X, const armpl_int_t incX,
                 const armpl_doublecomplex_t *beta, armpl_doublecomplex_t *Y,
                 const armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_zhemv*. It also exists with a native Fortran interface as *zhemv*.

3.5.129 cblas_zher

Syntax

```
#include "armpl.h"

void cblas_zher(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                const armpl_int_t N, const double alpha,
                const armpl_doublecomplex_t *X, const armpl_int_t incX,
                armpl_doublecomplex_t *A, const armpl_int_t lda);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_zher*. It also exists with a native Fortran interface as *zher*.

3.5.130 cblas_zher2

Syntax

```
#include "armpl.h"

void cblas_zher2(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const armpl_int_t N, const armpl_doublecomplex_t *alpha,
                 const armpl_doublecomplex_t *X, const armpl_int_t incX,
                 const armpl_doublecomplex_t *Y, const armpl_int_t incY,
                 armpl_doublecomplex_t *A, const armpl_int_t lda);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_zher2*. It also exists with a native Fortran interface as *zher2*.

3.5.131 cblas_zher2k

Syntax

```
#include "armpl.h"

void cblas_zher2k(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo,
                  const enum CBLAS_TRANSPOSE Trans, const armpl_int_t N,
                  const armpl_int_t K, const armpl_doublecomplex_t *alpha,
                  const armpl_doublecomplex_t *A, const armpl_int_t lda,
                  const armpl_doublecomplex_t *B, const armpl_int_t ldb,
                  const double beta, armpl_doublecomplex_t *C,
                  const armpl_int_t ldc);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_zher2k*. It also exists with a native Fortran interface as *zher2k*.

3.5.132 cblas_zherk

Syntax

```
#include "armpl.h"

void cblas_zherk(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE Trans, const armpl_int_t N,
                 const armpl_int_t K, const double alpha,
                 const armpl_doublecomplex_t *A, const armpl_int_t lda,
                 const double beta, armpl_doublecomplex_t *C,
                 const armpl_int_t ldc);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_zherk*. It also exists with a native Fortran interface as *zherk*.

3.5.133 cblas_zhpmv

Syntax

```
#include "armpl.h"

void cblas_zhpmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const armpl_int_t N, const armpl_doublecomplex_t *alpha,
                 const armpl_doublecomplex_t *Ap,
                 const armpl_doublecomplex_t *X, const armpl_int_t incX,
                 const armpl_doublecomplex_t *beta, armpl_doublecomplex_t *Y,
                 const armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_zhpmv*. It also exists with a native Fortran interface as *zhpmv*.

3.5.134 cblas_zhpr

Syntax

```
#include "armpl.h"

void cblas_zhpr(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                const armpl_int_t N, const double alpha,
                const armpl_doublecomplex_t *X, const armpl_int_t incX,
                armpl_doublecomplex_t *A);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_zhpr*. It also exists with a native Fortran interface as *zhpr*.

3.5.135 cblas_zhpr2

Syntax

```
#include "armpl.h"

void cblas_zhpr2(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const armpl_int_t N, const armpl_doublecomplex_t *alpha,
                 const armpl_doublecomplex_t *X, const armpl_int_t incX,
                 const armpl_doublecomplex_t *Y, const armpl_int_t incY,
                 armpl_doublecomplex_t *Ap);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_zhpr2*. It also exists with a native Fortran interface as *zhpr2*.

3.5.136 cblas_zscal

Syntax

```
#include "armpl.h"

void cblas_zscal(const armpl_int_t N, const armpl_doublecomplex_t *alpha,
                 armpl_doublecomplex_t *X, const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dscal*, *cblas_sscal* and *cblas_zscal*. It also exists with a native Fortran interface as *zscal*.

3.5.137 cblas_zswap

Syntax

```
#include "armpl.h"

void cblas_zswap(const armpl_int_t N, armpl_doublecomplex_t *X,
                 const armpl_int_t incX, armpl_doublecomplex_t *Y,
                 const armpl_int_t incY);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dswap*, *cblas_sswap* and *cblas_zswap*. It also exists with a native Fortran interface as *zswap*.

3.5.138 cblas_zsymm

Syntax

```
#include "armpl.h"

void cblas_zsymm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side,
                 const enum CBLAS_UPLO Uplo, const armpl_int_t M,
                 const armpl_int_t N, const armpl_doublecomplex_t *alpha,
                 const armpl_doublecomplex_t *A, const armpl_int_t lda,
                 const armpl_doublecomplex_t *B, const armpl_int_t ldb,
                 const armpl_doublecomplex_t *beta, armpl_doublecomplex_t *C,
                 const armpl_int_t ldc);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dsymm*, *cblas_ssymm* and *cblas_zsymm*. It also exists with a native Fortran interface as *zsymm*.

3.5.139 cblas_zsyr2k

Syntax

```
#include "armpl.h"

void cblas_zsyr2k(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo,
                  const enum CBLAS_TRANSPOSE Trans, const armpl_int_t N,
                  const armpl_int_t K, const armpl_doublecomplex_t *alpha,
                  const armpl_doublecomplex_t *A, const armpl_int_t lda,
                  const armpl_doublecomplex_t *B, const armpl_int_t ldb,
                  const armpl_doublecomplex_t *beta, armpl_doublecomplex_t *C,
                  const armpl_int_t ldc);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dsyr2k*, *cblas_ssyr2k* and *cblas_zsyr2k*. It also exists with a native Fortran interface as *zsyr2k*.

3.5.140 cblas_zsyrk

Syntax

```
#include "armpl.h"

void cblas_zsyrk(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE Trans, const armpl_int_t N,
                 const armpl_int_t K, const armpl_doublecomplex_t *alpha,
                 const armpl_doublecomplex_t *A, const armpl_int_t lda,
                 const armpl_doublecomplex_t *beta, armpl_doublecomplex_t *C,
                 const armpl_int_t ldc);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dsyrk*, *cblas_ssyrk* and *cblas_zsyrk*. It also exists with a native Fortran interface as *zsyrk*.

3.5.141 cblas_ztbmv

Syntax

```
#include "armpl.h"

void cblas_ztbmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE TransA,
                 const enum CBLAS_DIAG Diag, const armpl_int_t N,
                 const armpl_int_t K, const armpl_doublecomplex_t *A,
                 const armpl_int_t lda, armpl_doublecomplex_t *X,
                 const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dtbmv*, *cblas_stbmv* and *cblas_ztbmv*. It also exists with a native Fortran interface as *ztbmv*.

3.5.142 cblas_ztbsv

Syntax

```
#include "armpl.h"

void cblas_ztbsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE TransA,
                 const enum CBLAS_DIAG Diag, const armpl_int_t N,
                 const armpl_int_t K, const armpl_doublecomplex_t *A,
                 const armpl_int_t lda, armpl_doublecomplex_t *X,
                 const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dtbsv*, *cblas_stbsv* and *cblas_ztbsv*. It also exists with a native Fortran interface as *ztbsv*.

3.5.143 cblas_ztpmv

Syntax

```
#include "armpl.h"

void cblas_ztpmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE TransA,
                 const enum CBLAS_DIAG Diag, const armpl_int_t N,
                 const armpl_doublecomplex_t *Ap, armpl_doublecomplex_t *X,
                 const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dtpmv*, *cblas_stpmv* and *cblas_ztpmv*. It also exists with a native Fortran interface as *ztpmv*.

3.5.144 cblas_ztpsv

Syntax

```
#include "armpl.h"

void cblas_ztpsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE TransA,
                 const enum CBLAS_DIAG Diag, const armpl_int_t N,
                 const armpl_doublecomplex_t *Ap, armpl_doublecomplex_t *X,
                 const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dtpsv*, *cblas_stpsv* and *cblas_ztpsv*. It also exists with a native Fortran interface as *ztpsv*.

3.5.145 cblas_ztrmm

Syntax

```
#include "armpl.h"

void cblas_ztrmm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side,
                 const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE TransA,
                 const enum CBLAS_DIAG Diag, const armpl_int_t M,
                 const armpl_int_t N, const armpl_doublecomplex_t *alpha,
                 const armpl_doublecomplex_t *A, const armpl_int_t lda,
                 armpl_doublecomplex_t *B, const armpl_int_t ldb);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dtrmm*, *cblas_strmm* and *cblas_ztrmm*. It also exists with a native Fortran interface as *ztrmm*.

3.5.146 cblas_ztrmv

Syntax

```
#include "armpl.h"

void cblas_ztrmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE TransA,
                 const enum CBLAS_DIAG Diag, const armpl_int_t N,
                 const armpl_doublecomplex_t *A, const armpl_int_t lda,
                 armpl_doublecomplex_t *X, const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dtrmv*, *cblas_strmv* and *cblas_ztrmv*. It also exists with a native Fortran interface as *ztrmv*.

3.5.147 cblas_ztrsm

Syntax

```
#include "armpl.h"

void cblas_ztrsm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side,
                 const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE TransA,
                 const enum CBLAS_DIAG Diag, const armpl_int_t M,
                 const armpl_int_t N, const armpl_doublecomplex_t *alpha,
                 const armpl_doublecomplex_t *A, const armpl_int_t lda,
                 armpl_doublecomplex_t *B, const armpl_int_t ldb);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dtrsm*, *cblas_strsm* and *cblas_ztrsm*. It also exists with a native Fortran interface as *ztrsm*.

3.5.148 cblas_ztrsv

Syntax

```
#include "armpl.h"

void cblas_ztrsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo,
                 const enum CBLAS_TRANSPOSE TransA,
                 const enum CBLAS_DIAG Diag, const armpl_int_t N,
                 const armpl_doublecomplex_t *A, const armpl_int_t lda,
                 armpl_doublecomplex_t *X, const armpl_int_t incX);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_dtrsv*, *cblas_strsv* and *cblas_ztrsv*. It also exists with a native Fortran interface as *ztrsv*.

3.5.149 cblas_zwaxpby

Syntax

```
#include "armpl.h"

void cblas_zwaxpby(armpl_int_t N, const armpl_doublecomplex_t *alpha,
                   const armpl_doublecomplex_t *X, armpl_int_t incX,
                   const armpl_doublecomplex_t *beta,
                   armpl_doublecomplex_t *Y, armpl_int_t incY,
                   armpl_doublecomplex_t *W, armpl_int_t incw);
```

Parameters

CBLAS routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- All parameters marked `const CBLAS_LAYOUT` take a value of either `CblasRowMajor` or `CblasColMajor`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.
- All parameters marked `const CBLAS_TRANSPOSE` take a value of either `CblasNoTrans`, `CblasTrans` or `CblasConjTrans`, depending on whether the matrix in question in given in regular, transposed or conjugate form, respectively.

Related Information

For this routine in other precisions, please see *cblas_cwaxpby*, *cblas_dwaxpby* and *cblas_swaxpby*. It also exists with a native Fortran interface as *zwaxpby*.

3.6 BLAS Extensions

3.6.1 caxpby

`caxpby` performs the following vector update:

$y_i \leftarrow ax_i + by_i$ for $i = 0, n-1$.

Syntax

Fortran specification:

```
use armpl_library

subroutine caxpby(N, A, X, INCX, B, Y, INCY)
```

C specification:

```
#include "armpl.h"

void caxpby_(const armpl_int_t *n, const armpl_singlecomplex_t *alpha,
             const armpl_singlecomplex_t *x, const armpl_int_t *incx,
             const armpl_singlecomplex_t *beta, armpl_singlecomplex_t *y,
             const armpl_int_t *incy);
```

Parameters

N Input parameter.

N is INTEGER

The length of the vectors $\underline{x}$ and $\underline{y}$.

A Input parameter.

A is COMPLEX

The scalar a.

X Input parameter.

X is COMPLEX

X is an array, dimension *(1+(N-1)*ABS(INCX))*

Contains the elements of the vector $\underline{x}$ with the I^{th} element stored in X(1+(I-1)*ABS(INCX)).

INCX Input parameter.

INCX is INTEGER

INCX is the increment used to store successive elements of the Vector in X.

B Input parameter.

B is COMPLEX

The scalar b.

Y Input and output parameter.

Y is COMPLEX

Y is an array, dimension *(1+(N-1)*ABS(INCX))*

Contains the elements of the vector $\underline{x}$ with the I^{th} element stored in X(1+(I-1)*ABS(INCX)).

INCY Input parameter.

INCY is INTEGER

INCY is the increment used to store successive elements of the Vector in Y.

Related Information

For this routine in other precisions, please see *daxpby*, *saxpby* and *zaxpby*. It also exists with a native C interface as *cblas_caxpby*.

3.6.2 cgemm3m

cgemm3m performs one of the following matrix-matrix operations:

$$C \leftarrow \alpha op(A) op(B) + \beta C$$

where $op(X)$ is defined as one of:

$op(X) = X$

$op(X) = X^T$ (matrix transpose)

$op(X) = X^H$ (matrix conjugate transpose)

and α and β are scalars, $op(A)$ is an m by k matrix, $op(B)$ a k by n matrix and C is an m by n matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgemm3m(TRANSA, TRANSB, M, N, K ALPHA, A, LDA, B, LDB, BETA, C,
                   LDC)
```

C specification:

```
#include "armpl.h"

void cgemm3m_(const char *transa, const char *transb, const armpl_int_t *m,
              const armpl_int_t *n, const armpl_int_t *k,
              const armpl_singlecomplex_t *alpha,
              const armpl_singlecomplex_t *a, const armpl_int_t *lda,
              const armpl_singlecomplex_t *b, const armpl_int_t *ldb,
              const armpl_singlecomplex_t *beta, armpl_singlecomplex_t *c,
              const armpl_int_t *ldc, ... );
```

Parameters

TRANSA Input parameter

TRANSA is CHARACTER*1

On input:

- If TRANSA = 'n' or 'N' $op(A) = A$;
- If TRANSA = 't' or 'T' $op(A) = A^T$;
- If TRANSA = 'c' or 'C' $op(A) = A^H$.

TRANSB Input parameter

TRANSB is CHARACTER*1

On input:

- If TRANSB = 'n' or 'N' $op(B) = B$;
- If TRANSB = 't' or 'T' $op(B) = B^T$;
- If TRANSB = 'c' or 'C' $op(B) = B^H$.

M Input parameter

M is INTEGER

The number of rows of the matrices $p(A)$ and C.

Constraint $M \geq 0$.

N Input parameter

N is INTEGER

On input: The number of columns of the matrices $op(B)$ and :math‘C‘.

Constraint $N \geq 0$.

K Input parameter

K is INTEGER

On input: The number of columns of the matrix $op(A)$ and rows of the matrix $op(B)$.

Constraint $K \geq 0$.

ALPHA Input parameter

ALPHA is COMPLEX

On input: The value of the scalar α.

A Input parameter

A is COMPLEX

On input: An array of dimension (LDA, KA) containing the matrix A, where if *TRANSA = ‘n’ or ‘N’* then KA = K* and the matrix is held in the leading M by K part of the array, otherwise *KA = M* and the matrix is held in the leading K by M part of the array.

LDA Input parameter

LDA is INTEGER **LDA**

On input: The leading dimension of the array *A* as declared in the calling subprogram.

Constraint If *TRANSA = ‘n’ or ‘N’* then $LDA \geq max(1, M)$, otherwise $LDA \geq max(1, K)$.

B Input parameter

B is COMPLEX

On input: An array of dimension (LDB, KB) containing the matrix B, where if *TRANSB = ‘n’ or ‘N’* then *KB = N* and the matrix is held in the leading K by N part of the array, otherwise *KB = K* and the matrix is held in the leading *N* by *K* part of the array.

LDB Input parameter

LDB is INTEGER

On input: The leading dimension of the array *B* as declared in the calling subprogram.

Constraint If *TRANSB = ‘n’ or ‘N’* then $LDB \geq max(1, K)$, otherwise $LDB \geq max(1, N)$.

BETA Input parameter

`BETA` is ``COMPLEX

On input: The value of the scalar β.

C Input and output parameter

C is COMPLEX

On input: An array of dimension (LDC, N) containing the matrix C, in the leading *M* by *N* part of the array. Note that if BETA = 0 then the matrix *C* need not be initialised on input.

On output: The array is overwritten with the result of the operation: $\alpha op(A) op(B) + \beta C$.

LDC Input parameter

LDC is INTEGER

On input: The leading dimension of the array C as declared in the calling subprogram.

Constraint $LDC \geq max(1, M)$.

Related Information

For this routine in other precisions, please see *zgemm3m*. It also exists with a native C interface as *cblas_cgemm3m*.

3.6.3 cwaxpby

cwaxpby performs the following vector operation:

$w_i \leftarrow ax_i + by_i$ for $i = 0, n - 1$.

Syntax

Fortran specification:

```
use armpl_library

subroutine cwaxpby(N, A, X, INCX, B, Y, INCY, W, INCW)
```

C specification:

```
#include "armpl.h"

void cwaxpby_(const armpl_int_t *n, const armpl_singlecomplex_t *alpha,
              const armpl_singlecomplex_t *x, const armpl_int_t *incx,
              const armpl_singlecomplex_t *beta, armpl_singlecomplex_t *y,
              const armpl_int_t *incy, armpl_singlecomplex_t *w,
              const armpl_int_t *incw);
```

Parameters

N Input parameter.

N is INTEGER

The length of the vectors $\underline{x}$ and $\underline{y}$.

A Input parameter.

A is COMPLEX

The scalar a.

X Input parameter.

X is COMPLEX

X is an array, dimension (1+(N-1)*ABS(INCX))

Contains the elements of the vector $\underline{x}$ with the I^{th} element stored in X(1+(I-1)*ABS(INCX)).

INCX Input parameter.

INCX is INTEGER

INCX is the increment used to store successive elements of the vector in X.

B Input parameter.

B is COMPLEX

The scalar b.

Y Input parameter.

Y is COMPLEX

Y is an array, dimension (1+(N-1)*ABS(INCX))

Contains the elements of the vector $\underline{x}$ with the I^{th} element stored in X(1+(I-1)*ABS(INCX)).

INCY Input parameter.

INCY is INTEGER

INCY is the increment used to store successive elements of the vector in Y.

W Output parameter.

W is COMPLEX

W is an array, dimension (1+(N-1)*ABS(INCX))

Contains the elements of the vector $\underline{x}$ with the I^{th} element stored in X(1+(I-1)*ABS(INCX)).

INCW Input parameter.

INCW is INTEGER

INCW is the increment used to store successive elements of the vector in W.

Related Information

For this routine in other precisions, please see *caxpby*, *saxpby* and *daxpby*. It also exists with a native C interface as *cblas_zaxpby*.

3.6.4 daxpby

daxpby performs the following vector update:

$y_i \leftarrow ax_i + by_i$ for $i = 0, n-1$.

Syntax

Fortran specification:

```
use armpl_library

subroutine daxpby(N, A, X, INCX, B, Y, INCY)
```

C specification:

```
#include "armpl.h"

void daxpby_(const armpl_int_t *n, const double *alpha,
             const double *x, const armpl_int_t *incx,
             const double *beta, double *y,
             const armpl_int_t *incy);
```

Parameters

N Input parameter.

N is INTEGER

The length of the vectors $\underline{x}$ and $\underline{y}$.

A Input parameter.

A is DOUBLE PRECISION

The scalar a.

X Input parameter.

X is DOUBLE PRECISION

X is an array, dimension *(1+(N-1)*ABS(INCX))*

Contains the elements of the vector $\underline{x}$ with the I^{th} element stored in X(1+(I-1)*ABS(INCX)).

INCX Input parameter.

INCX is INTEGER

INCX is the increment used to store successive elements of the Vector in X.

B Input parameter.

B is DOUBLE PRECISION

The scalar b.

Y Input and output parameter.

Y is DOUBLE PRECISION

Y is an array, dimension *(1+(N-1)*ABS(INCX))*

Contains the elements of the vector $\underline{x}$ with the I^{th} element stored in X(1+(I-1)*ABS(INCX)).

INCY Input parameter.

INCY is INTEGER

INCY is the increment used to store successive elements of the Vector in Y.

Related Information

For this routine in other precisions, please see *caxpby*, *saxpby* and *zaxpby*. It also exists with a native C interface as *cblas_caxpby*.

3.6.5 dwaxpby

dwaxpby performs the following vector operation:

$w_i \leftarrow ax_i + by_i$ for $i = 0, n - 1$.

Syntax

Fortran specification:

```
use armpl_library

subroutine dwaxpby(N, A, X, INCX, B, Y, INCY, W, INCW)
```

C specification:

```
#include "armpl.h"

void dwaxpby_(const armpl_int_t *n, const armpl_doublecomplex_t *alpha,
              const armpl_doublecomplex_t *x, const armpl_int_t *incx,
              const armpl_doublecomplex_t *beta, armpl_doublecomplex_t *y,
              const armpl_int_t *incy, armpl_doublecomplex_t *w,
              const armpl_int_t *incw);
```

Parameters

N Input parameter.

N is INTEGER

The length of the vectors $\underline{x}$ and $\underline{y}$.

A Input parameter.

A is DOUBLE PRECISION

The scalar a.

X Input parameter.

X is DOUBLE PRECISION

X is an array, dimension (1+(N-1)*ABS(INCX))

Contains the elements of the vector $\underline{x}$ with the I^{th} element stored in X(1+(I-1)*ABS(INCX)).

INCX Input parameter.

INCX is INTEGER

INCX is the increment used to store successive elements of the vector in X.

B Input parameter.

B is DOUBLE PRECISION

The scalar b.

Y Input parameter.

Y is DOUBLE PRECISION

Y is an array, dimension (1+(N-1)*ABS(INCX))

Contains the elements of the vector $\underline{x}$ with the I^{th} element stored in X(1+(I-1)*ABS(INCX)).

INCY Input parameter.

INCY is INTEGER

INCY is the increment used to store successive elements of the vector in Y.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (1+(N-1)*ABS(INCX))

Contains the elements of the vector $\underline{x}$ with the I^{th} element stored in X(1+(I-1)*ABS(INCX)).

INCW Input parameter.

INCW is INTEGER

INCW is the increment used to store successive elements of the vector in W.

Related Information

For this routine in other precisions, please see *caxpby*, *saxpby* and *daxpby*. It also exists with a native C interface as *cblas_zaxpby*.

3.6.6 hgemm

`hgemm` performs one of the matrix-matrix operations

```
C := alpha*op( A )*op( B ) + beta*C,
```

where op(X) is one of

```
op( X ) = X   or   op( X ) = X**T,
```

alpha and beta are scalars, and A, B and C are matrices, with op(A) an m by k matrix, op(B) a k by n matrix and C an m by n matrix.

Syntax

C specification:

```
#include "armpl.h"

void hgemm_(const char *transa, const char *transb, const armpl_int_t *m,
            const armpl_int_t *n, const armpl_int_t *k, const __fp16 *alpha,
            const __fp16 *A, const armpl_int_t *lda, const __fp16 *B,
            const armpl_int_t *ldb, const __fp16 *beta, __fp16 *C,
            const armpl_int_t *ldc, ... );
```

Parameters

`transa` Input parameter.

`transa` is `const char *`

On entry, `transa` specifies the form of op(`A`) to be used in the matrix multiplication as follows:

`transa` = 'N' or 'n', op(`A`) = A.

`transa` = 'T' or 't', op(`A`) = A^T .

`transa` = 'C' or 'c', op(`A`) = A^T .

`transb` Input parameter.

`transb` is `const char *`

On entry, `transb` specifies the form of op(`B`) to be used in the matrix multiplication as follows:

`transb` = 'N' or 'n', op(`B`) = B.

`transb` = 'T' or 't', op(`B`) = B^T .

`transb` = 'C' or 'c', op(`B`) = B^T .

`m` Input parameter.

`m` is `armpl_int_t *`

On entry, `m` specifies the number of rows of the matrix op(`A`) and of the matrix `C`. `m` must be at least zero.

n Input parameter.

n is armpl_int_t *

On entry, n specifies the number of columns of the matrix op(B) and the number of columns of the matrix C. n must be at least zero.

k Input parameter.

k is armpl_int_t *

On entry, k specifies the number of columns of the matrix op(A) and the number of rows of the matrix op(B). k must be at least zero.

alpha Input parameter.

alpha is __fp16 *

On entry, alpha specifies the scalar alpha.

A Input parameter.

A is __fp16 *

A is an array, dimension (lda, ka), where ka is. k when transa = 'N' or 'n', and is m otherwise. Before entry with transa = 'N' or 'n', the leading m by k part of the array A must contain the matrix A, otherwise the leading k by m part of the array A must contain the matrix A.

lda Input parameter.

lda is armpl_int_t *

On entry, lda specifies the first dimension of A as declared in the calling (sub) program. When transa = 'N' or 'n' then lda must be at least max(1, m), otherwise lda must be at least max(1, k).

B Input parameter.

B is __fp16 *

B is an array, dimension (ldb, kb), where kb is. n when transb = 'N' or 'n', and is k otherwise. Before entry with transb = 'N' or 'n', the leading k by n part of the array B must contain the matrix B, otherwise the leading n by k part of the array B must contain the matrix B.

ldb Input parameter.

ldb is armpl_int_t *

On entry, ldb specifies the first dimension of B as declared in the calling (sub) program. When transb = 'N' or 'n' then ldb must be at least max(1, k), otherwise ldb must be at least max(1, n).

beta Input parameter.

beta is __fp16 *

On entry, beta specifies the scalar beta. When beta is supplied as zero then C need not be set on input.

C Input and output parameter.

C is __fp16 *

C is an array, dimension (ldc, N). Before entry, the leading m by n part of the array C must contain the matrix C, except when beta is zero, in which case C need not be set on entry. On exit, the array C is overwritten by the m by n matrix (alpha*op(A)*op(B) + beta* C).

ldc Input parameter.

ldc is armpl_int_t *

On entry, ldc specifies the first dimension of C as declared in the calling (sub) program. ldc must be at least max(1, m).

Related Information

For this routine in other precisions, please see *cgemm*, *dgemm*, *sgemm* and *zgemm*.

3.6.7 saxpby

saxpby performs the following vector update:

$y_i \leftarrow ax_i + by_i$ for $i = 0, n-1$.

Syntax

Fortran specification:

```
use armpl_library

subroutine saxpby(N, A, X, INCX, B, Y, INCY)
```

C specification:

```
#include "armpl.h"

void saxpby_(const armpl_int_t *n, const real *alpha,
             const real *x, const armpl_int_t *incx,
             const real *beta, real *y,
             const armpl_int_t *incy);
```

Parameters

N Input parameter.

N is INTEGER

The length of the vectors $\underline{x}$ and $\underline{y}$.

A Input parameter.

A is DOUBLE PRECISION

The scalar a.

X Input parameter.

X is DOUBLE PRECISION

X is an array, dimension *(1+(N-1)*ABS(INCX))*

Contains the elements of the vector $\underline{x}$ with the I^{th} element stored in X(1+(I-1)*ABS(INCX)).

INCX Input parameter.

INCX is INTEGER

INCX is the increment used to store successive elements of the Vector in X.

B Input parameter.

B is DOUBLE PRECISION

The scalar b.

Y Input and output parameter.

Y is DOUBLE PRECISION

Y is an array, dimension *(1+(N-1)*ABS(INCX))*

Contains the elements of the vector $\underline{x}$ with the I^{th} element stored in X(1+(I-1)*ABS(INCX)).

INCY Input parameter.

INCY is INTEGER

INCY is the increment used to store successive elements of the Vector in Y.

Related Information

For this routine in other precisions, please see *caxpby*, *daxpby* and *zaxpby*. It also exists with a native C interface as *cblas_saxpby*.

3.6.8 swaxpby

swaxpby performs the following vector operation:

$w_i \leftarrow ax_i + by_i$ for $i = 0, n-1$.

Syntax

Fortran specification:

```
use armpl_library

subroutine swaxpby(N, A, X, INCX, B, Y, INCY, W, INCW)
```

C specification:

```
#include "armpl.h"

void swaxpby_(const armpl_int_t *n, const armpl_doublecomplex_t *alpha,
              const armpl_doublecomplex_t *x, const armpl_int_t *incx,
              const armpl_doublecomplex_t *beta, armpl_doublecomplex_t *y,
              const armpl_int_t *incy, armpl_doublecomplex_t *w,
              const armpl_int_t *incw);
```

Parameters

N Input parameter.

N is INTEGER

The length of the vectors $\underline{x}$ and $\underline{y}$.

A Input parameter.

A is REAL

The scalar a.

X Input parameter.

X is REAL

X is an array, dimension (1+(N-1)*ABS(INCX))

Contains the elements of the vector $\underline{x}$ with the I^{th} element stored in X(1+(I-1)*ABS(INCX)).

INCX Input parameter.

INCX is INTEGER

INCX is the increment used to store successive elements of the vector in X.

B Input parameter.

B is REAL

The scalar b.

Y Input parameter.

Y is REAL

Y is an array, dimension (1+(N-1)*ABS(INCX))

Contains the elements of the vector $\underline{x}$ with the I^{th} element stored in X(1+(I-1)*ABS(INCX)).

INCY Input parameter.

INCY is INTEGER

INCY is the increment used to store successive elements of the vector in Y.

W Output parameter.

W is REAL

W is an array, dimension (1+(N-1)*ABS(INCX))

Contains the elements of the vector $\underline{x}$ with the I^{th} element stored in X(1+(I-1)*ABS(INCX)).

INCW Input parameter.

INCW is INTEGER

INCW is the increment used to store successive elements of the vector in W.

Related Information

For this routine in other precisions, please see *caxpby*, *saxpby* and *daxpby*. It also exists with a native C interface as *cblas_zaxpby*.

3.6.9 zaxpby

zaxpby performs the following vector update:

$y_i \leftarrow ax_i + by_i$ for $i = 0, n - 1$.

Syntax

Fortran specification:

```
use armpl_library

subroutine zaxpby(N, A, X, INCX, B, Y, INCY)
```

C specification:

```
#include "armpl.h"

void zaxpby_(const armpl_int_t *n, const armpl_doublecomplex_t *alpha,
             const armpl_doublecomplex_t *x, const armpl_int_t *incx,
             const armpl_doublecomplex_t *beta, armpl_doublecomplex_t *y,
             const armpl_int_t *incy);
```

Parameters

N Input parameter.

N is INTEGER

The length of the vectors $\underline{x}$ and $\underline{y}$.

A Input parameter.

A is COMPLEX*16

The scalar a.

X Input parameter.

X is COMPLEX*16

X is an array, dimension *(1+(N-1)*ABS(INCX))*

Contains the elements of the vector $\underline{x}$ with the I^{th} element stored in X(1+(I-1)*ABS(INCX)).

INCX Input parameter.

INCX is INTEGER

INCX is the increment used to store successive elements of the Vector in X.

B Input parameter.

B is COMPLEX*16

The scalar b.

Y Input and output parameter.

Y is COMPLEX*16

Y is an array, dimension *(1+(N-1)*ABS(INCX))*

Contains the elements of the vector $\underline{x}$ with the I^{th} element stored in X(1+(I-1)*ABS(INCX)).

INCY Input parameter.

INCY is INTEGER

INCY is the increment used to store successive elements of the Vector in Y.

Related Information

For this routine in other precisions, please see *caxpby*, *saxpby* and *daxpby*. It also exists with a native C interface as *cblas_zaxpby*.

3.6.10 zgemm3m

zgemm3m performs one of the following matrix-matrix operations:

$$C \leftarrow \alpha op(A) op(B) + \beta C$$

where $op(X)$ is defined as one of:

$op(X) = X$

$op(X) = X^T$ (matrix transpose)

$op(X) = X^H$ (matrix conjugate transpose)

and α and β are scalars, $op(A)$ is an m by k matrix, $op(B)$ a k by n matrix and C is an m by n matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgemm3m(TRANSA, TRANSB, M, N, K ALPHA, A, LDA, B, LDB, BETA, C,
                   LDC)
```

C specification:

```
#include "armpl.h"

void zgemm3m_(const char *transa, const char *transb, const armpl_int_t *m,
              const armpl_int_t *n, const armpl_int_t *k,
              const armpl_doublecomplex_t *alpha,
              const armpl_doublecomplex_t *a, const armpl_int_t *lda,
              const armpl_doublecomplex_t *b, const armpl_int_t *ldb,
              const armpl_doublecomplex_t *beta, armpl_doublecomplex_t *c,
              const armpl_int_t *ldc, ... );
```

Parameters

TRANSA Input parameter

TRANSA is CHARACTER*1

On input:

- If TRANSA = 'n' or 'N' $op(A) = A$;
- If TRANSA = 't' or 'T' $op(A) = A^T$;
- If TRANSA = 'c' or 'C' $op(A) = A^H$.

TRANSB Input parameter

TRANSB is CHARACTER*1

On input:

- If TRANSB = 'n' or 'N' $op(B) = B$;
- If TRANSB = 't' or 'T' $op(B) = B^T$;
- If TRANSB = 'c' or 'C' $op(B) = B^H$.

M Input parameter

M is INTEGER

The number of rows of the matrices $p(A)$ and C.

Constraint $M \geq 0$.

N Input parameter

N is INTEGER

On input: The number of columns of the matrices $op(B)$ and :math'C'.

Constraint $N \geq 0$.

K Input parameter

K is INTEGER

On input: The number of columns of the matrix $op(A)$ and rows of the matrix $op(B)$.

Constraint $K \geq 0$.

ALPHA Input parameter

ALPHA is COMPLEX*16

On input: The value of the scalar α.

A Input parameter

A is COMPLEX*16

On input: An array of dimension (LDA, KA) containing the matrix A, where if *TRANSA = 'n' or 'N'* then KA = K* and the matrix is held in the leading M by K part of the array, otherwise *KA = M* and the matrix is held in the leading K by M part of the array.

LDA Input parameter

LDA is INTEGER **LDA**

On input: The leading dimension of the array *A* as declared in the calling subprogram.

Constraint If *TRANSA = 'n' or 'N'* then $LDA \geq max(1, M)$, otherwise $LDA \geq max(1, K)$.

B Input parameter

B is COMPLEX*16

On input: An array of dimension (LDB, KB) containing the matrix B, where if *TRANSB = 'n' or 'N'* then *KB = N* and the matrix is held in the leading K by N part of the array, otherwise *KB = K* and the matrix is held in the leading *N* by *K* part of the array.

LDB Input parameter

LDB is INTEGER

On input: The leading dimension of the array *B* as declared in the calling subprogram.

Constraint If *TRANSB = 'n' or 'N'* then $LDB \geq max(1, K)$, otherwise $LDB \geq max(1, N)$.

BETA Input parameter

`BETA` is ``COMPLEX*16

On input: The value of the scalar β.

C Input and output parameter

C is COMPLEX*16

On input: An array of dimension (LDC, N) containing the matrix C, in the leading *M* by *N* part of the array. Note that if BETA = 0 then the matrix *C* need not be initialised on input.

On output: The array is overwritten with the result of the operation: $\alpha op(A) op(B) + \beta C$.

LDC Input parameter

LDC is INTEGER

On input: The leading dimension of the array C as declared in the calling subprogram.

Constraint $LDC \geq max(1, M)$.

Related Information

For this routine in other precisions, please see *cgemm3m*. It also exists with a native C interface as *cblas_zgemm3m*.

3.6.11 zwaxpby

zwaxpby performs the following vector operation:

$w_i \leftarrow ax_i + by_i$ for $i = 0, n - 1$.

Syntax

Fortran specification:

```
use armpl_library

subroutine zwaxpby(N, A, X, INCX, B, Y, INCY, W, INCW)
```

C specification:

```
#include "armpl.h"

void zwaxpby_(const armpl_int_t *n, const armpl_doublecomplex_t *alpha,
              const armpl_doublecomplex_t *x, const armpl_int_t *incx,
              const armpl_doublecomplex_t *beta, armpl_doublecomplex_t *y,
              const armpl_int_t *incy, armpl_doublecomplex_t *w,
              const armpl_int_t *incw);
```

Parameters

N Input parameter.

N is INTEGER

The length of the vectors $\underline{x}$ and $\underline{y}$.

A Input parameter.

A is COMPLEX*16

The scalar a.

X Input parameter.

X is COMPLEX*16

X is an array, dimension (1+(N-1)*ABS(INCX))

Contains the elements of the vector $\underline{x}$ with the I^{th} element stored in X(1+(I-1)*ABS(INCX)).

INCX Input parameter.

INCX is INTEGER

INCX is the increment used to store successive elements of the vector in X.

B Input parameter.

B is COMPLEX*16

The scalar b.

Y Input parameter.

Y is COMPLEX*16

Y is an array, dimension (1+(N-1)*ABS(INCX))

Contains the elements of the vector $\underline{x}$ with the I^{th} element stored in X(1+(I-1)*ABS(INCX)).

INCY Input parameter.

INCY is INTEGER

INCY is the increment used to store successive elements of the vector in Y.

W Output parameter.

W is COMPLEX*16

W is an array, dimension (1+(N-1)*ABS(INCX))

Contains the elements of the vector $\underline{x}$ with the I^{th} element stored in `X(1+(I-1)*ABS(INCX))`.

`INCW` Input parameter.

`INCW` is `INTEGER`

`INCW` is the increment used to store successive elements of the vector in `W`.

Related Information

For this routine in other precisions, please see *caxpby*, *saxpby* and *daxpby*. It also exists with a native C interface as *cblas_zaxpby*.

3.6.12 sgemm_batch

`sgemm_batch` computes a batch of `TOTAL_SIZE` *sgemm* problems which are split into `GROUP_COUNT` groups, where problems in a group share the same transpose options, problem dimensions, array leading dimensions and scalars α and β.

This routine implements the following operation:

```
   TOTAL_SIZE = 0
   PID = 1
   DO 10 J = 1, GROUP_COUNT
      DO 20 I = 1, GROUP_SIZE(J)
         CALL SGEMM(TRANSA(J),TRANSB(J),M(J),N(J),K(J),ALPHA(J),
  &                 A(PID),LDA(J),B(PID),LDB(J),BETA(J),C(PID),LDC(J))
         PID = PID + 1
20    CONTINUE
      TOTAL_SIZE = TOTAL_SIZE + GROUP_SIZE(J)
10 CONTINUE
```

Note that the above also defines the value of `TOTAL_SIZE`, which is not required as a routine argument, but is used in the documentation below; it is the sum of the `GROUP_SIZE` array.

sgemm is the standard BLAS single precision complex matrix-matrix multiplication routine, and users are referred to its own documentation.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgemm_batch(TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA,
                       C, LDC, GROUP_COUNT, GROUP_SIZE)
```

C specification:

```
#include "armpl.h"

void sgemm_batch_(const char *transa, const char *transb,
                  const armpl_int_t *m, const armpl_int_t *n,
                  const armpl_int_t *k, const float *alpha,
                  const float *const *a,
                  const armpl_int_t *lda,
                  const float *const *b,
                  const armpl_int_t *ldb, const float *beta,
                  float *const *c, const armpl_int_t *ldc,
                  const armpl_int_t *group_count,
                  const armpl_int_t *group_size, ... );
```

Parameters

TRANSA Input parameter.

TRANSA is CHARACTER*1

TRANSA is an array, dimension (GROUP_COUNT)

- If TRANSA(J) = 'n' or 'N' then for the J^{th} group, for all matrices A, $op(A) = A$;
- If TRANSA(J) = 't' or 'T' or 'c' or 'C'vthen for the J^{th} group, for all matrices A, $op(A) = A^T$ (matrix transpose).

TRANSB Input parameter.

TRANSB is CHARACTER*1

TRANSB is an array, dimension (GROUP_COUNT)

- If TRANSB(J) = 'n' or 'N' then for the J^{th} group, for all matrices B, $op(B) = B$;
- If TRANSB(J) = 't' or 'T' or 'c' or 'C' then for the J^{th} group, for all matrices B, $op(B) = B^T$ (matrix transpose).

M Input parameter.

M is INTEGER

M is an array, dimension (GROUP_COUNT)

For all matrices A in the J^{th} group, M(J) specifies the number of rows of $op(A)$ and C.

Constraint: M(J) ≥ 0 for J=1,GROUP_COUNT.

N Input parameter.

N is INTEGER

N is an array, dimension (GROUP_COUNT)

For all matrices B and C in the J^{th} group, N(J) specifies the number of columns of $op(B)$ and C.

Constraint: N(J) ≥ 0 for J=1,GROUP_COUNT.

K Input parameter.

K is INTEGER

K is an array, dimension (GROUP_COUNT)

For all matrices A and B in the J^{th} group, K(J) specifies the number of columns of op(A) ` and rows of op(B).

Constraint: $K(J) \geq 0$ for J=1,GROUP_COUNT.

ALPHA Input parameter.

ALPHA is REAL

ALPHA is an array, dimension (GROUP_COUNT)

ALPHA(J) gives the value of the scalar α for the J^{th} group.

A Input parameter.

A is REAL

A is an array, dimension (TOTAL_SIZE)

An array of pointers to TOTAL_SIZE input matrices A.

LDA Input parameter.

LDA is INTEGER

LDA is an array, dimension (GROUP_COUNT)

LDA(J) gives the leading dimension of all A matrices in the J^{th} group as declared in the calling subprogram.

Constraint: If TRANSA(J) = 'n' or 'N' then LDA(J) $\geq$ max(1,M(J)) , otherwise LDA(J) $\geq$ max(1, K(J)) for J=1,GROUP_COUNT.

B Input parameter.

B is REAL

B is an array, dimension (TOTAL_SIZE)

An array of pointers to TOTAL_SIZE input matrices *B*.

LDB Input parameter.

LDB is INTEGER

LDB is an array, dimension (GROUP_COUNT)

LDB(J) gives the leading dimension of all B matrices in the J^{th} group as declared in the calling subprogram.

Constraint: If TRANSB(J) = 'n' or 'N' then LDA(J) $\geq$ max(1,K(J)) , otherwise LDA(J) $\geq$ max(1, N(J)) for J=1,GROUP_COUNT.

BETA Input parameter.

BETA is REAL

BETA is an array, dimension (GROUP_COUNT)

BETA(J) gives the value of the scalar β for the J^{th} group.

C Input and output parameter.

C is REAL

C is an array, dimension (TOTAL_SIZE)

An array of pointers to TOTAL_SIZE input matrices C.

LDC Input parameter.

LDC is INTEGER

LDC is an array, dimension (GROUP_COUNT)

LDC(J) gives the leading dimension of all C matrices in the J^{th} group as declared in the calling subprogram.

Constraint: LDC(J) $\geq$ max(1,M(J)) for J=1,GROUP_COUNT.

GROUP_COUNT Input parameter.

GROUP_COUNT is INTEGER

Specifies the number of groups.

Constraint: GROUP_COUNT ≥ 0.

GROUP_SIZE Input parameter.

GROUP_SIZE is INTEGER

GROUP_SIZE is an array, dimension (GROUP_COUNT)

On input: GROUP_SIZE(J) specifies the number of *sgemm* problems in the J^{th} group.

Constraint: GROUP_SIZE(J) ≥ 0 for J=1,GROUP_COUNT.

3.6.13 dgemm_batch

dgemm_batch computes a batch of TOTAL_SIZE *dgemm* problems which are split into GROUP_COUNT groups, where problems in a group share the same transpose options, problem dimensions, array leading dimensions and scalars α and β.

This routine implements the following operation:

```
   TOTAL_SIZE = 0
   PID = 1
   DO 10 J = 1, GROUP_COUNT
      DO 20 I = 1, GROUP_SIZE(J)
         CALL DGEMM(TRANSA(J),TRANSB(J),M(J),N(J),K(J),ALPHA(J),
  &                  A(PID),LDA(J),B(PID),LDB(J),BETA(J),C(PID),LDC(J))
         PID = PID + 1
20    CONTINUE
      TOTAL_SIZE = TOTAL_SIZE + GROUP_SIZE(J)
10 CONTINUE
```

Note that the above also defines the value of TOTAL_SIZE, which is not required as a routine argument, but is used in the documentation below; it is the sum of the GROUP_SIZE array.

dgemm is the standard BLAS single precision complex matrix-matrix multiplication routine, and users are referred to its own documentation.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgemm_batch(TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA,
                       C, LDC, GROUP_COUNT, GROUP_SIZE)
```

C specification:

```
#include "armpl.h"

void dgemm_batch_(const char *transa, const char *transb,
                  const armpl_int_t *m, const armpl_int_t *n,
                  const armpl_int_t *k, const double *alpha,
                  const double *const *a,
                  const armpl_int_t *lda,
                  const double *const *b,
                  const armpl_int_t *ldb, const double *beta,
                  double *const *c, const armpl_int_t *ldc,
                  const armpl_int_t *group_count,
                  const armpl_int_t *group_size, ... );
```

Parameters

TRANSA Input parameter.

TRANSA is CHARACTER*1

TRANSA is an array, dimension (GROUP_COUNT)

- If TRANSA(J) = 'n' or 'N' then for the J^{th} group, for all matrices A, $op(A) = A$;
- If TRANSA(J) = 't' or 'T' or 'c' or 'C'vthen for the J^{th} group, for all matrices A, $op(A) = A^T$ (matrix transpose).

TRANSB Input parameter.

TRANSB is CHARACTER*1

TRANSB is an array, dimension (GROUP_COUNT)

DOUBLE PRECISION

- If TRANSB(J) = 'n' or 'N' then for the J^{th} group, for all matrices B, $op(B) = B$;
- If TRANSB(J) = 't' or 'T' or 'c' or 'C' then for the J^{th} group, for all matrices B, $op(B) = B^T$ (matrix transpose).

M Input parameter.

M is INTEGER

M is an array, dimension (GROUP_COUNT)

For all matrices A in the J^{th} group, M(J) specifies the number of rows of $op(A)$ and C.

Constraint: M(J) ≥ 0 for J=1,GROUP_COUNT.

N Input parameter.

N is INTEGER

N is an array, dimension (GROUP_COUNT)

For all matrices B and C in the J^{th} group, N(J) specifies the number of columns of $op(B)$ and C.

Constraint: N(J) ≥ 0 for J=1,GROUP_COUNT.

K Input parameter.

K is INTEGER

K is an array, dimension (GROUP_COUNT)

For all matrices A and B in the J^{th} group, K(J) specifies the number of columns of op(A)` and rows of op(B).

Constraint: $K(J) \geq 0$ for J=1,GROUP_COUNT.

ALPHA Input parameter.

ALPHA is DOUBLE PRECISION

ALPHA is an array, dimension (GROUP_COUNT)

ALPHA(J) gives the value of the scalar α for the J^{th} group.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (TOTAL_SIZE)

An array of pointers to TOTAL_SIZE input matrices *A*.

LDA Input parameter.

LDA is INTEGER

LDA is an array, dimension (GROUP_COUNT)

LDA(J) gives the leading dimension of all A matrices in the J^{th} group as declared in the calling subprogram.

Constraint: If TRANSA(J) = 'n' or 'N' then LDA(J) $\geq$ max(1,M(J)) , otherwise LDA(J) $\geq$ max(1, K(J)) for J=1,GROUP_COUNT.

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (TOTAL_SIZE)

An array of pointers to TOTAL_SIZE input matrices *B*.

LDB Input parameter.

LDB is INTEGER

LDB is an array, dimension (GROUP_COUNT)

LDB(J) gives the leading dimension of all B matrices in the J^{th} group as declared in the calling subprogram.

Constraint: If TRANSB(J) = 'n' or 'N' then LDA(J) $\geq$ max(1,K(J)) , otherwise LDA(J) $\geq$ max(1, N(J)) for J=1,GROUP_COUNT.

BETA Input parameter.

BETA is DOUBLE PRECISION

BETA is an array, dimension (GROUP_COUNT)

BETA(J) gives the value of the scalar β for the J^{th} group.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (TOTAL_SIZE)

An array of pointers to TOTAL_SIZE input matrices C.

LDC Input parameter.

LDC is INTEGER

LDC is an array, dimension (GROUP_COUNT)

LDC(J) gives the leading dimension of all C matrices in the J^{th} group as declared in the calling subprogram.

Constraint: LDC(J) $\geq$ max(1,M(J)) for J=1,GROUP_COUNT.

GROUP_COUNT Input parameter.

GROUP_COUNT is INTEGER

Specifies the number of groups.

Constraint: GROUP_COUNT ≥ 0.

GROUP_SIZE Input parameter.

GROUP_SIZE is INTEGER

GROUP_SIZE is an array, dimension (GROUP_COUNT)

On input: GROUP_SIZE(J) specifies the number of *dgemm* problems in the J^{th} group.

Constraint: GROUP_SIZE(J) ≥ 0 for J=1,GROUP_COUNT.

3.6.14 cgemm_batch

cgemm_batch computes a batch of TOTAL_SIZE *cgemm* problems which are split into GROUP_COUNT groups, where problems in a group share the same transpose options, problem dimensions, array leading dimensions and scalars α and β.

This routine implements the following operation:

```
   TOTAL_SIZE = 0
   PID = 1
   DO 10 J = 1, GROUP_COUNT
      DO 20 I = 1, GROUP_SIZE(J)
         CALL CGEMM(TRANSA(J),TRANSB(J),M(J),N(J),K(J),ALPHA(J),
  &                 A(PID),LDA(J),B(PID),LDB(J),BETA(J),C(PID),LDC(J))
```

(continues on next page)

(continued from previous page)

```
        PID = PID + 1
20    CONTINUE
      TOTAL_SIZE = TOTAL_SIZE + GROUP_SIZE(J)
10 CONTINUE
```

Note that the above also defines the value of TOTAL_SIZE, which is not required as a routine argument, but is used in the documentation below; it is the sum of the GROUP_SIZE array.

cgemm is the standard BLAS single precision complex matrix-matrix multiplication routine, and users are referred to its own documentation.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgemm_batch(TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA,
                       C, LDC, GROUP_COUNT, GROUP_SIZE)
```

C specification:

```
#include "armpl.h"

void cgemm_batch_(const char *transa, const char *transb,
                  const armpl_int_t *m, const armpl_int_t *n,
                  const armpl_int_t *k, const armpl_singlecomplex_t *alpha,
                  const armpl_singlecomplex_t *const *a,
                  const armpl_int_t *lda,
                  const armpl_singlecomplex_t *const *b,
                  const armpl_int_t *ldb, const armpl_singlecomplex_t *beta,
                  armpl_singlecomplex_t *const *c, const armpl_int_t *ldc,
                  const armpl_int_t *group_count,
                  const armpl_int_t *group_size, ... );
```

Parameters

TRANSA Input parameter.

TRANSA is CHARACTER*1

TRANSA is an array, dimension (GROUP_COUNT)

- If TRANSA(J) = 'n' or 'N' then for the J^{th} group, for all matrices A, $op(A) = A$;
- If TRANSA(J) = 't' or 'T' then for the J^{th} group, for all matrices A, $op(A) = A^T$ (matrix transpose).
- If TRANSA(J) = 'c' or 'C' then for the J^{th} group, for all matrices A, $op(A) = A^H$ (matrix conjugate transpose).

TRANSB Input parameter.

TRANSB is CHARACTER*1

TRANSB is an array, dimension (GROUP_COUNT)

- If TRANSB(J) = 'n' or 'N' then for the J^{th} group, for all matrices B, $op(B) = B$;
- If TRANSB(J) = 't' or 'T' then for the J^{th} group, for all matrices B, $op(B) = B^T$ (matrix transpose).
- If TRANSB(J) = 'c' or 'C' then for the J^{th} group, for all matrices B, $op(B) = B^H$ (matrix conjugate transpose).

M Input parameter.

M is INTEGER

M is an array, dimension (GROUP_COUNT)

For all matrices A in the J^{th} group, M(J) specifies the number of rows of $op(A)$ and C.

Constraint: M(J) ≥ 0 for J=1,GROUP_COUNT.

N Input parameter.

N is INTEGER

N is an array, dimension (GROUP_COUNT)

For all matrices B and C in the J^{th} group, N(J) specifies the number of columns of $op(B)$ and C.

Constraint: N(J) ≥ 0 for J=1,GROUP_COUNT.

K Input parameter.

K is INTEGER

K is an array, dimension (GROUP_COUNT)

For all matrices A and B in the J^{th} group, K(J) specifies the number of columns of op(A)` and rows of op(B).

Constraint: $K(J) \geq 0$ for J=1,GROUP_COUNT.

ALPHA Input parameter.

ALPHA is COMPLEX

ALPHA is an array, dimension (GROUP_COUNT)

ALPHA(J) gives the value of the scalar α for the J^{th} group.

A Input parameter.

A is COMPLEX

A is an array, dimension (TOTAL_SIZE)

An array of pointers to TOTAL_SIZE input matrices *A*.

LDA Input parameter.

LDA is INTEGER

LDA is an array, dimension (GROUP_COUNT)

LDA(J) gives the leading dimension of all A matrices in the J^{th} group as declared in the calling subprogram.

Constraint: If TRANSA(J) = 'n' or 'N' then LDA(J) $\geq$ max(1,M(J)) , otherwise LDA(J) $\geq$ max(1, K(J)) for J=1,GROUP_COUNT.

B Input parameter.

B is COMPLEX

B is an array, dimension (TOTAL_SIZE)

An array of pointers to TOTAL_SIZE input matrices *B*.

LDB Input parameter.

LDB is INTEGER

LDB is an array, dimension (GROUP_COUNT)

LDB(J) gives the leading dimension of all B matrices in the J^{th} group as declared in the calling subprogram.

Constraint: If TRANSB(J) = 'n' or 'N' then LDA(J) $\geq$ max(1,K(J)) , otherwise LDA(J) $\geq$ max(1, N(J)) for J=1,GROUP_COUNT.

BETA Input parameter.

BETA is COMPLEX

BETA is an array, dimension (GROUP_COUNT)

BETA(J) gives the value of the scalar β for the J^{th} group.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (TOTAL_SIZE)

An array of pointers to TOTAL_SIZE input matrices C.

LDC Input parameter.

LDC is INTEGER

LDC is an array, dimension (GROUP_COUNT)

LDC(J) gives the leading dimension of all C matrices in the J^{th} group as declared in the calling subprogram.

Constraint: LDC(J) $\geq$ max(1,M(J)) for J=1,GROUP_COUNT.

GROUP_COUNT Input parameter.

GROUP_COUNT is INTEGER

Specifies the number of groups.

Constraint: GROUP_COUNT ≥ 0.

GROUP_SIZE Input parameter.

GROUP_SIZE is INTEGER

GROUP_SIZE is an array, dimension (GROUP_COUNT)

On input: GROUP_SIZE(J) specifies the number of *zgemm* problems in the J^{th} group.

Constraint: GROUP_SIZE(J) ≥ 0 for J=1,GROUP_COUNT.

3.6.15 zgemm_batch

zgemm_batch computes a batch of TOTAL_SIZE *zgemm* problems which are split into GROUP_COUNT groups, where problems in a group share the same transpose options, problem dimensions, array leading dimensions and scalars α and β.

This routine implements the following operation:

```
   TOTAL_SIZE = 0
   PID = 1
   DO 10 J = 1, GROUP_COUNT
      DO 20 I = 1, GROUP_SIZE(J)
         CALL ZGEMM(TRANSA(J),TRANSB(J),M(J),N(J),K(J),ALPHA(J),
  &                 A(PID),LDA(J),B(PID),LDB(J),BETA(J),C(PID),LDC(J))
         PID = PID + 1
20    CONTINUE
      TOTAL_SIZE = TOTAL_SIZE + GROUP_SIZE(J)
10 CONTINUE
```

Note that the above also defines the value of TOTAL_SIZE, which is not required as a routine argument, but is used in the documentation below; it is the sum of the GROUP_SIZE array.

zgemm is the standard BLAS single precision complex matrix-matrix multiplication routine, and users are referred to its own documentation.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgemm_batch(TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA,
                       C, LDC, GROUP_COUNT, GROUP_SIZE)
```

C specification:

```
#include "armpl.h"

void zgemm_batch_(const char *transa, const char *transb,
                  const armpl_int_t *m, const armpl_int_t *n,
                  const armpl_int_t *k, const armpl_doublecomplex_t *alpha,
                  const armpl_doublecomplex_t *const *a,
                  const armpl_int_t *lda,
                  const armpl_doublecomplex_t *const *b,
                  const armpl_int_t *ldb, const armpl_doublecomplex_t *beta,
                  armpl_doublecomplex_t *const *c, const armpl_int_t *ldc,
                  const armpl_int_t *group_count,
                  const armpl_int_t *group_size, ... );
```

Parameters

TRANSA Input parameter.

TRANSA is CHARACTER*1

TRANSA is an array, dimension (GROUP_COUNT)

- If TRANSA(J) = 'n' or 'N' then for the J^{th} group, for all matrices A, $op(A) = A$;
- If TRANSA(J) = 't' or 'T' then for the J^{th} group, for all matrices A, $op(A) = A^T$ (matrix transpose).
- If TRANSA(J) = 'c' or 'C' then for the J^{th} group, for all matrices A, $op(A) = A^H$ (matrix conjugate transpose).

TRANSB Input parameter.

TRANSB is CHARACTER*1

TRANSB is an array, dimension (GROUP_COUNT)

- If TRANSB(J) = 'n' or 'N' then for the J^{th} group, for all matrices B, $op(B) = B$;
- If TRANSB(J) = 't' or 'T' then for the J^{th} group, for all matrices B, $op(B) = B^T$ (matrix transpose).
- If TRANSB(J) = 'c' or 'C' then for the J^{th} group, for all matrices B, $op(B) = B^H$ (matrix conjugate transpose).

M Input parameter.

M is INTEGER

M is an array, dimension (GROUP_COUNT)

For all matrices A in the J^{th} group, M(J) specifies the number of rows of $op(A)$ and C.

Constraint: M(J) ≥ 0 for J=1,GROUP_COUNT.

N Input parameter.

N is INTEGER

N is an array, dimension (GROUP_COUNT)

For all matrices B and C in the J^{th} group, N(J) specifies the number of columns of $op(B)$ and C.

Constraint: `N(J)` ≥ 0 for `J=1,GROUP_COUNT`.

K Input parameter.

`K` is `INTEGER`

`K` is an array, dimension `(GROUP_COUNT)`

For all matrices `A` and `B` in the J^{th} group, `K(J)` specifies the number of columns of `op(A)` ` and rows of `op(B)`.

Constraint: $K(J) \geq 0$ for `J=1,GROUP_COUNT`.

ALPHA Input parameter.

`ALPHA` is `COMPLEX*16`

`ALPHA` is an array, dimension `(GROUP_COUNT)`

`ALPHA(J)` gives the value of the scalar α for the J^{th} group.

A Input parameter.

`A` is `COMPLEX*16`

`A` is an array, dimension `(TOTAL_SIZE)`

An array of pointers to `TOTAL_SIZE` input matrices *A*.

LDA Input parameter.

`LDA` is `INTEGER`

`LDA` is an array, dimension `(GROUP_COUNT)`

`LDA(J)` gives the leading dimension of all `A` matrices in the J^{th} group as declared in the calling subprogram.

Constraint: If `TRANSA(J)` = 'n' or 'N' then `LDA(J)` $\geq$ `max(1,M(J))` , otherwise `LDA(J)` $\geq$ `max(1, K(J))` for `J=1,GROUP_COUNT`.

B Input parameter.

`B` is `COMPLEX*16`

`B` is an array, dimension `(TOTAL_SIZE)`

An array of pointers to `TOTAL_SIZE` input matrices *B*.

LDB Input parameter.

`LDB` is `INTEGER`

`LDB` is an array, dimension `(GROUP_COUNT)`

`LDB(J)` gives the leading dimension of all `B` matrices in the J^{th} group as declared in the calling subprogram.

Constraint: If `TRANSB(J)` = 'n' or 'N' then `LDA(J)` $\geq$ `max(1,K(J))` , otherwise `LDA(J)` $\geq$ `max(1, N(J))` for `J=1,GROUP_COUNT`.

BETA Input parameter.

`BETA` is `COMPLEX*16`

`BETA` is an array, dimension `(GROUP_COUNT)`

`BETA(J)` gives the value of the scalar β for the J^{th} group.

C Input and output parameter.

`C` is `COMPLEX*16`

`C` is an array, dimension `(TOTAL_SIZE)`

An array of pointers to `TOTAL_SIZE` input matrices `C`.

LDC Input parameter.

LDC is INTEGER

LDC is an array, dimension (GROUP_COUNT)

LDC(J) gives the leading dimension of all C matrices in the J^{th} group as declared in the calling subprogram.

Constraint: LDC(J) $\geq$ max(1,M(J))` for J=1,GROUP_COUNT.

GROUP_COUNT Input parameter.

GROUP_COUNT is INTEGER

Specifies the number of groups.

Constraint: GROUP_COUNT ≥ 0.

GROUP_SIZE Input parameter.

GROUP_SIZE is INTEGER

GROUP_SIZE is an array, dimension (GROUP_COUNT)

On input: GROUP_SIZE(J) specifies the number of *zgemm* problems in the J^{th} group.

Constraint: GROUP_SIZE(J) ≥ 0 for J=1,GROUP_COUNT.

In addition to the standard BLAS interfaces, Arm Performance Libraries also supports some commonly used extensions. We document here the Fortran interfaces, but the equivalent CBLAS-like interfaces are supported and prototypes are provided in armpl.h.

CHAPTER

FOUR

LAPACK LINEAR ALGEBRA PACKAGE

4.1 Introduction to LAPACK

LAPACK ([4]) is a library of FORTRAN 77 subroutines for solving commonly occurring problems in numerical linear algebra. LAPACK components can solve systems of linear equations, linear least squares problems, eigenvalue problems and singular value problems. Dense and banded matrices are provided for, but not general sparse matrices. In all areas, similar functionality is provided for real and complex matrices.

LAPACK routines are written so that as much as possible of the computations is performed by calls to the BLAS. The efficiency of LAPACK routines depends, in large part, on the efficiency of the BLAS being called. Block algorithms are employed wherever possible to maximize the use of calls to level 3 BLAS, which generally run faster than lower level BLAS due to the high number of operations per memory access.

The performance of some of the LAPACK routines has been further improved by reworking the computational algorithms. Some of the LAPACK routines contained in Arm Performance Libraries are therefore based on code that is different from the LAPACK sources available in the public domain. In all these cases the algorithmic and numerical properties of the original LAPACK routines have been strictly preserved. Furthermore, key LAPACK routines have been treated using OpenMP to take advantage of multiple processors when running on SMP machines. Your application will automatically benefit when you link with the OpenMP versions of Arm Performance Libraries.

4.2 Reference sources for LAPACK

The LAPACK homepage can be accessed on the World Wide Web via the URL address:

http://www.netlib.org/lapack/

The on-line version of the Lapack User's Guide, Third Edition ([4]) is available from this homepage, or directly using the URL:

http://www.netlib.org/lapack/lug/index.html

The standard source code is available for download from netlib:

http://www.netlib.org/lapack/lapack.tgz

A list of known problems, bugs, and compiler errors for LAPACK, as well as an errata list for the LAPACK User's Guide ([4]), is maintained on netlib

http://www.netlib.org/lapack/release_notes

A LAPACK FAQ (Frequently Asked Questions) file can also be accessed via the LAPACK homepage

http://www.netlib.org/lapack/faq.html

For C users, Arm Performance Libraries includes the LAPACKE C interfaces to Fortran LAPACK. Names of these routines are similar to the Fortran routines except they are prefixed by the string `LAPACKE_`. For example, the Fortran LAPACK routine `dgetrf` becomes `LAPACKE_dgetrf` in C. Other differences between Fortran BLAS and CBLAS interfaces:

- Scalar input arguments are passed by value in LAPACKE interfaces. FORTRAN interfaces pass all arguments (except for character string *length* arguments that are normally hidden from FORTRAN programmers) by reference.
- Unlike FORTRAN, C has no native *complex* data type. Arm Performance Libraries C routines which operate on complex data use the types `armpl_singlecomplex_t` and `armpl_doublecomplex_t` defined in `armpl.h` for single and double precision computations respectively.
- As with Fortran, all LAPACKE array arguments are required to be stored in contiguous memory. However, all Fortran LAPACK routines which take one or more matrices (2D arrays) as arguments have an extra parameter in the LAPACKE interface, which appears before all other parameters. This parameter is of integer type and is named `matrix_order`, and it can take one of the values `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, where `LAPACK_ROW_MAJOR` and `LAPACK_COL_MAJOR` are integers defined in `armpl.h`

  The value `LAPACK_COL_MAJOR` indicates that elements within any column of each array are contiguous in memory while elements within array rows are offset by a constant stride. This is the usual Fortran storage order. Such strides are given by "leading dimension" arguments (such as `LDA`) in the Fortran and C interfaces.

  The value `LAPACK_ROW_MAJOR` indicates that elements within any row of each 2D array are contiguous in memory while elements within array columns are offset by a constant stride such as `LDA`.
- Fortran's `character` type arguments are passed as values of the C `char` type.
- In general, Fortran "workspace" type arguments (with names such as `WORK`, `RWORK` and `CWORK`) do not appear in the C interfaces - workspace memory is allocated internally. However, LAPACKE also includes variants which *do* have the workspace arguments - these variants have the string `_work` appended to the function name, for example `LAPACKE_dgetrf_work`.

See the `armpl.h` header file to find prototypes for all LAPACKE functions. Note that the functions may also be called from a C++ program if you include `armpl.h`.

4.3 LAPACK matrix factorization routines

4.3.1 cgbtrf

`cgbtrf` computes an LU factorization of a complex m-by-n band matrix A using partial pivoting with row interchanges.

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgbtrf(M, N, KL, KU, AB, LDAB, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void cgbtrf_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *kl, const armpl_int_t *ku,
             armpl_singlecomplex_t *ab, const armpl_int_t *ldab,
             armpl_int_t *ipiv, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

AB Input and output parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows KL+1 to 2*KL+KU+1; rows 1 to KL of the array need not be set. The j-th column of A is stored in the j-th column of the array AB as follows: AB(kl+ku+1+i-j,j) = A(i,j) for max(1,j-ku)<=i<=min(m,j+kl)

On exit, details of the factorization: U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1. See below for further details.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= 2*KL+KU+1.

IPIV Output parameter.

IPIV is INTEGER array, dimension (min(M, N))

The pivot indices; for 1 <= i <= min(M, N), row i of the matrix was interchanged with row IPIV(i).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = +i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *dgbtrf*, *sgbtrf* and *zgbtrf*. It also exists with a native C interface as *LAPACKE_cgbtrf*.

4.3.2 cgelq

cgelq computes a LQ factorization of an M-by-N matrix A.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgelq(M, N, A, LDA, T, TSIZE, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgelq_(const armpl_int_t *m, const armpl_int_t *n,
            armpl_singlecomplex_t *a, const armpl_int_t *lda,
            armpl_singlecomplex_t *t, const armpl_int_t *tsize,
            armpl_singlecomplex_t *work, const armpl_int_t *lwork,
            armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and below the diagonal of the array contain the M-by-min(M, N) lower trapezoidal matrix L (L is lower triangular if M <= N); the elements above the diagonal are used to store part of the data structure to represent Q.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

T Output parameter.

T is COMPLEX

T is an array, dimension (MAX(5, TSIZE)). On exit, if INFO = 0, T(1) returns optimal (or either minimal or optimal, if query is assumed) TSIZE. See TSIZE for details. Remaining T contains part of the data structure used to represent Q. If one wants to apply or construct Q, then one needs to keep T (in addition to A) and pass it to further subroutines.

TSIZE Input parameter.

TSIZE is INTEGER

If TSIZE >= 5, the dimension of the array T. If TSIZE = -1 or -2, then a workspace query is assumed. The routine only calculates the sizes of the T and WORK arrays, returns these values as the first entries of the T and WORK arrays, and no error message related to T or WORK is issued by XERBLA. If TSIZE = -1, the routine calculates optimal size of T for the optimum performance and returns this value in T(1). If TSIZE = -2, the routine calculates minimal size of T and returns this value in T(1).

WORK Output parameter.

(workspace) COMPLEX

(workspace) is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) contains optimal (or either minimal or optimal, if query was assumed) LWORK. See LWORK for details.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If LWORK = -1 or -2, then a workspace query is assumed. The routine only calculates the sizes of the T and WORK arrays, returns these values as the first entries of the T and WORK arrays, and no error message related to T or WORK is issued by XERBLA. If LWORK = -1, the routine calculates optimal size of WORK for the optimal performance and returns this value in WORK(1). If LWORK = -2, the routine calculates minimal size of WORK and returns this value in WORK(1).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dgelq*, *sgelq* and *zgelq*. It also exists with a native C interface as *LAPACKE_cgelq*.

4.3.3 cgelqf

cgelqf computes an LQ factorization of a complex M-by-N matrix A: A = L * Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgelqf(M, N, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgelqf_(const armpl_int_t *m, const armpl_int_t *n,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             armpl_singlecomplex_t *tau, armpl_singlecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and below the diagonal of the array contain the m-by-min(m,n) lower trapezoidal matrix L (L is lower triangular if m <= n); the elements above the diagonal, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is COMPLEX

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, M). For optimum performance LWORK >= M*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dgelqf*, *sgelqf* and *zgelqf*. It also exists with a native C interface as *LAPACKE_cgelqf*.

4.3.4 cgelqt

cgelqt computes a blocked LQ factorization of a complex M-by-N matrix A using the compact WY representation of Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgelqt(M, N, MB, A, LDA, T, LDT, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgelqt_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *mb, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_singlecomplex_t *t,
             const armpl_int_t *ldt, armpl_singlecomplex_t *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

MB Input parameter.

MB is INTEGER

The block size to be used in the blocked QR. MIN(M, N) >= MB >= 1.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and below the diagonal of the array contain the M-by-MIN(M, N) lower trapezoidal matrix L (L is lower triangular if M <= N); the elements above the diagonal are the rows of V.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

T Output parameter.

T is COMPLEX

T is an array, dimension (LDT,MIN(M, N)). The upper triangular block reflectors stored in compact form as a sequence of upper triangular blocks. See below for further details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= MB.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MB*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dgelqt*, *sgelqt* and *zgelqt*.

4.3.5 cgemlq

cgemlq overwrites the general real M-by-N matrix C with SIDE = 'L' SIDE = 'R' TRANS = 'N': Q * C C * Q TRANS = 'C': Q^H * C C * Q^H where Q is a complex unitary matrix defined as the product of blocked elementary reflectors computed by short wide LQ factorization (CGELQ)

Syntax

Fortran specification:

```
use armpl_library

subroutine cgemlq(SIDE, TRANS, M, N, K, A, LDA, T, TSIZE, C, LDC, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void cgemlq_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const armpl_singlecomplex_t *t, const armpl_int_t *tsize,
             armpl_singlecomplex_t *c, const armpl_int_t *ldc,
             armpl_singlecomplex_t *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'T': Transpose, apply Q^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >=0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension. (LDA, M) if SIDE = 'L', (LDA, N) if SIDE = 'R' Part of the data structure to represent Q as returned by CGELQ.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, K).

T Input parameter.

T is COMPLEX

T is an array, dimension (MAX(5, TSIZE)).. Part of the data structure to represent Q as returned by CGELQ.

TSIZE Input parameter.

TSIZE is INTEGER

The dimension of the array T. TSIZE >= 5.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

(workspace) COMPLEX

(workspace) is an array, dimension (MAX(1, LWORK)) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If LWORK = -1, then a workspace query is assumed. The routine only calculates the size of the WORK array, returns this value as WORK(1), and no error message related to WORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dgemlq*, *sgemlq* and *zgemlq*. It also exists with a native C interface as *LAPACKE_cgemlq*.

4.3.6 cgemlqt

cgemlqt overwrites the general real M-by-N matrix C with SIDE = 'L' SIDE = 'R' TRANS = 'N': Q C C Q TRANS = 'C': Q^H C C Q^H where Q is a complex orthogonal matrix defined as the product of K elementary reflectors: Q = H(1) H(2) . . . H(K) = I - V T V^H generated using the compact WY representation as returned by CGELQT. Q is of order M if SIDE = 'L' and of order N if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgemlqt(SIDE, TRANS, M, N, K, MB, V, LDV, T, LDT, C, LDC, WORK,
                   INFO)
```

C specification:

```
#include "armpl.h"

void cgemlqt_(const char *side, const char *trans, const armpl_int_t *m,
              const armpl_int_t *n, const armpl_int_t *k,
              const armpl_int_t *mb, const armpl_singlecomplex_t *v,
              const armpl_int_t *ldv, const armpl_singlecomplex_t *t,
              const armpl_int_t *ldt, armpl_singlecomplex_t *c,
              const armpl_int_t *ldc, armpl_singlecomplex_t *work,
              armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left; = 'R': apply Q or Q^H from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'C': Transpose, apply Q^H .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

MB Input parameter.

MB is INTEGER

The block size used for the storage of T. K >= MB >= 1. This must be the same value of MB used to generate T in DGELQT.

V Input parameter.

V is COMPLEX

V is an array, dimension. (LDV, M) if SIDE = 'L', (LDV, N) if SIDE = 'R' The i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by DGELQT in the first K rows of its array argument A.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. LDV >= max(1, K).

T Input parameter.

T is COMPLEX

T is an array, dimension (LDT, K). The upper triangular factors of the block reflectors as returned by DGELQT, stored as a MB-by-K matrix.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= MB.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q C, Q^H C, C Q^H or C Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX array. The dimension of

WORK is N* MB if SIDE = 'L', or M* MB if SIDE = 'R'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dgemlqt*, *sgemlqt* and *zgemlqt*.

4.3.7 cgemqr

cgemqr overwrites the general real M-by-N matrix C with SIDE = 'L' SIDE = 'R' TRANS = 'N': Q * C C * Q TRANS = 'T': Q^H * C C * Q^H where Q is a complex unitary matrix defined as the product of blocked elementary reflectors computed by tall skinny QR factorization (CGEQR)

Syntax

Fortran specification:

```
use armpl_library

subroutine cgemqr(SIDE, TRANS, M, N, K, A, LDA, T, TSIZE, C, LDC, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void cgemqr_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const armpl_singlecomplex_t *t, const armpl_int_t *tsize,
             armpl_singlecomplex_t *c, const armpl_int_t *ldc,
             armpl_singlecomplex_t *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'T': Transpose, apply Q^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >=0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, K). Part of the data structure to represent Q as returned by CGEQR.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDA >= max(1, M); if SIDE = 'R', LDA >= max(1, N).

T Input parameter.

T is COMPLEX

T is an array, dimension (MAX(5, TSIZE)).. Part of the data structure to represent Q as returned by CGEQR.

TSIZE Input parameter.

TSIZE is INTEGER

The dimension of the array T. TSIZE >= 5.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

(workspace) COMPLEX

(workspace) is an array, dimension (MAX(1, LWORK)) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If LWORK = -1, then a workspace query is assumed. The routine only calculates the size of the WORK array, returns this value as WORK(1), and no error message related to WORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dgemqr*, *sgemqr* and *zgemqr*. It also exists with a native C interface as *LAPACKE_cgemqr*.

4.3.8 cgemqrt

cgemqrt overwrites the general complex M-by-N matrix C with

```
SIDE = 'L'      SIDE = 'R'
```

TRANS = 'N': Q C C Q TRANS = 'C': Q^H C C Q^H

where Q is a complex orthogonal matrix defined as the product of K elementary reflectors:

```
Q = H(1) H(2) . . . H(K) = I - V T V**H
```

generated using the compact WY representation as returned by CGEQRT.

Q is of order M if SIDE = 'L' and of order N if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgemqrt(SIDE, TRANS, M, N, K, NB, V, LDV, T, LDT, C, LDC, WORK,
                   INFO)
```

C specification:

```
#include "armpl.h"

void cgemqrt_(const char *side, const char *trans, const armpl_int_t *m,
              const armpl_int_t *n, const armpl_int_t *k,
              const armpl_int_t *nb, const armpl_singlecomplex_t *v,
              const armpl_int_t *ldv, const armpl_singlecomplex_t *t,
              const armpl_int_t *ldt, armpl_singlecomplex_t *c,
              const armpl_int_t *ldc, armpl_singlecomplex_t *work,
              armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left; = 'R': apply Q or Q^H from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'C': Transpose, apply Q^H .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

NB Input parameter.

NB is INTEGER

The block size used for the storage of T. K >= NB >= 1. This must be the same value of NB used to generate T in CGEQRT.

V Input parameter.

V is COMPLEX

V is an array, dimension (LDV, K). The i-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by CGEQRT in the first K columns of its array argument A.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. If SIDE = 'L', LDA >= max(1, M); if SIDE = 'R', LDA >= max(1, N).

T Input parameter.

T is COMPLEX

T is an array, dimension (LDT, K). The upper triangular factors of the block reflectors as returned by CGEQRT, stored as a NB-by-N matrix.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= NB.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q C, Q^H C, C Q^H or C Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX array. The dimension of WORK is

N* NB if SIDE = 'L', or M* NB if SIDE = 'R'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dgemqrt*, *sgemqrt* and *zgemqrt*. It also exists with a native C interface as *LAPACKE_cgemqrt*.

4.3.9 cgeqlf

cgeqlf computes a QL factorization of a complex M-by-N matrix A: A = Q * L.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgeqlf(M, N, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgeqlf_(const armpl_int_t *m, const armpl_int_t *n,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             armpl_singlecomplex_t *tau, armpl_singlecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, if m >= n, the lower triangle of the subarray A(m-n+1:m,1:n) contains the N-by-N lower triangular matrix L; if m <= n, the elements on and below the (n-m)-th superdiagonal contain the M-by-N lower trapezoidal matrix L; the remaining elements, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is COMPLEX

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N). For optimum performance LWORK >= N*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dgeqlf*, *sgeqlf* and *zgeqlf*. It also exists with a native C interface as *LAPACKE_cgeqlf*.

4.3.10 cgeqp3

cgeqp3 computes a QR factorization with column pivoting of a matrix A: A*P = Q*R using Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgeqp3(M, N, A, LDA, JPVT, TAU, WORK, LWORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgeqp3_(const armpl_int_t *m, const armpl_int_t *n,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             armpl_int_t *jpvt, armpl_singlecomplex_t *tau,
             armpl_singlecomplex_t *work, const armpl_int_t *lwork,
             float *rwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the upper triangle of the array contains the min(M, N)-by-N upper trapezoidal matrix R; the elements below the diagonal, together with the array TAU, represent the unitary matrix Q as a product of min(M, N) elementary reflectors.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

JPVT Input and output parameter.

JPVT is INTEGER array, dimension (N)

On entry, if JPVT(J).ne.0, the J-th column of A is permuted to the front of A*P (a leading column); if JPVT(J)=0, the J-th column of A is a free column. On exit, if JPVT(J)=K, then the J-th column of A*P was the the K-th column of A.

TAU Output parameter.

TAU is COMPLEX

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO=0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= N+1. For optimal performance LWORK >= (N+1)*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *dgeqp3*, *sgeqp3* and *zgeqp3*. It also exists with a native C interface as *LAPACKE_cgeqp3*.

4.3.11 cgeqr

cgeqr computes a QR factorization of an M-by-N matrix A.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgeqr(M, N, A, LDA, T, TSIZE, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgeqr_(const armpl_int_t *m, const armpl_int_t *n,
            armpl_singlecomplex_t *a, const armpl_int_t *lda,
            armpl_singlecomplex_t *t, const armpl_int_t *tsize,
            armpl_singlecomplex_t *work, const armpl_int_t *lwork,
            armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the min(M, N)-by-N upper trapezoidal matrix R (R is upper triangular if M >= N); the elements below the diagonal are used to store part of the data structure to represent Q.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

T Output parameter.

T is COMPLEX

T is an array, dimension (MAX(5, TSIZE)). On exit, if INFO = 0, T(1) returns optimal (or either minimal or optimal, if query is assumed) TSIZE. See TSIZE for details. Remaining T contains part of the data structure used to represent Q. If one wants to apply or construct Q, then one needs to keep T (in addition to A) and pass it to further subroutines.

TSIZE Input parameter.

TSIZE is INTEGER

If TSIZE >= 5, the dimension of the array T. If TSIZE = -1 or -2, then a workspace query is assumed. The routine only calculates the sizes of the T and WORK arrays, returns these values as the first entries of the T and WORK arrays, and no error message related to T or WORK is issued by XERBLA. If TSIZE = -1, the routine calculates optimal size of T for the optimum performance and returns this value in T(1). If TSIZE = -2, the routine calculates minimal size of T and returns this value in T(1).

WORK Output parameter.

(workspace) COMPLEX

(workspace) is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) contains optimal (or either minimal or optimal, if query was assumed) LWORK. See LWORK for details.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If LWORK = -1 or -2, then a workspace query is assumed. The routine only calculates the sizes of the T and WORK arrays, returns these values as the first entries of the T and WORK arrays, and no error message related to T or WORK is issued by XERBLA. If LWORK = -1, the routine calculates optimal size of WORK for the optimal performance and returns this value in WORK(1). If LWORK = -2, the routine calculates minimal size of WORK and returns this value in WORK(1).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dgeqr*, *sgeqr* and *zgeqr*. It also exists with a native C interface as *LAPACKE_cgeqr*.

4.3.12 cgeqrf

cgeqrf computes a QR factorization of a complex M-by-N matrix A: A = Q * R.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgeqrf(M, N, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgeqrf_(const armpl_int_t *m, const armpl_int_t *n,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             armpl_singlecomplex_t *tau, armpl_singlecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the min(M, N)-by-N upper trapezoidal matrix R (R is upper triangular if m >= n); the elements below the diagonal, with the array TAU, represent the unitary matrix Q as a product of min(m,n) elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is COMPLEX

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N). For optimum performance LWORK >= N*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dgeqrf*, *sgeqrf* and *zgeqrf*. It also exists with a native C interface as *LAPACKE_cgeqrf*.

4.3.13 cgeqrfp

cgeqrfp computes a QR factorization of a complex M-by-N matrix A: A = Q * R. The diagonal entries of R are real and nonnegative.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgeqrfp(M, N, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgeqrfp_(const armpl_int_t *m, const armpl_int_t *n,
              armpl_singlecomplex_t *a, const armpl_int_t *lda,
              armpl_singlecomplex_t *tau, armpl_singlecomplex_t *work,
              const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the min(M, N)-by-N upper trapezoidal matrix R (R is upper triangular if m >= n). The diagonal entries of R are real and nonnegative; the elements below the diagonal, with the array TAU, represent the unitary matrix Q as a product of min(m,n) elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is COMPLEX

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N). For optimum performance LWORK >= N*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dgeqrfp*, *sgeqrfp* and *zgeqrfp*. It also exists with a native C interface as *LAPACKE_cgeqrfp*.

4.3.14 cgeqrt

cgeqrt computes a blocked QR factorization of a complex M-by-N matrix A using the compact WY representation of Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgeqrt(M, N, NB, A, LDA, T, LDT, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgeqrt_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *nb, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_singlecomplex_t *t,
             const armpl_int_t *ldt, armpl_singlecomplex_t *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

NB Input parameter.

NB is INTEGER

The block size to be used in the blocked QR. MIN(M, N) >= NB >= 1.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the min(M, N)-by-N upper trapezoidal matrix R (R is upper triangular if M >= N); the elements below the diagonal are the columns of V.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

T Output parameter.

T is COMPLEX

T is an array, dimension (LDT,MIN(M, N)). The upper triangular block reflectors stored in compact form as a sequence of upper triangular blocks. See below for further details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= NB.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (NB*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dgeqrt*, *sgeqrt* and *zgeqrt*. It also exists with a native C interface as *LAPACKE_cgeqrt*.

4.3.15 cgerqf

cgerqf computes an RQ factorization of a complex M-by-N matrix A: A = R * Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgerqf(M, N, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgerqf_(const armpl_int_t *m, const armpl_int_t *n,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             armpl_singlecomplex_t *tau, armpl_singlecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, if m <= n, the upper triangle of the subarray A(1:m,n-m+1:n) contains the M-by-M upper triangular matrix R; if m >= n, the elements on and above the (m-n)-th subdiagonal contain the M-by-N upper trapezoidal matrix R; the remaining elements, with the array TAU, represent the unitary matrix Q as a product of min(m,n) elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is COMPLEX

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, M). For optimum performance LWORK >= M*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dgerqf*, *sgerqf* and *zgerqf*. It also exists with a native C interface as *LAPACKE_cgerqf*.

4.3.16 cgetrf

cgetrf computes an LU factorization of a general M-by-N matrix A using partial pivoting with row interchanges.

The factorization has the form

```
A = P * L * U
```

where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if m > n), and U is upper triangular (upper trapezoidal if m < n).

This is the right-looking Level 3 BLAS version of the algorithm.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgetrf(M, N, A, LDA, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void cgetrf_(const armpl_int_t *m, const armpl_int_t *n,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             armpl_int_t *ipiv, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the M-by-N matrix to be factored. On exit, the factors L and U from the factorization A = P*L*U; the unit diagonal elements of L are not stored.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

IPIV Output parameter.

IPIV is INTEGER array, dimension (min(M, N))

The pivot indices; for 1 <= i <= min(M, N), row i of the matrix was interchanged with row IPIV(i).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *dgetrf*, *sgetrf* and *zgetrf*. It also exists with a native C interface as *LAPACKE_cgetrf*.

4.3.17 cgetrf2

cgetrf2 computes an LU factorization of a general M-by-N matrix A using partial pivoting with row interchanges.

The factorization has the form

```
A = P * L * U
```

where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if m > n), and U is upper triangular (upper trapezoidal if m < n).

This is the recursive version of the algorithm. It divides the matrix into four submatrices:

```
    [  A11 | A12  ]  where A11 is n1 by n1 and A22 is n2 by n2
A = [ -----|----- ]  with n1 = min(m,n)/2
    [  A21 | A22  ]       n2 = n-n1

                                  [ A11 ]
```

The subroutine calls itself to factor [—],

```
                      [ A12 ]
[ A12 ]
```

do the swaps on [—], solve A12, update A22,

```
[ A22 ]
```

then calls itself to factor A22 and do the swaps on A21.

Syntax

Fortran specification:

```
use armpl_library

recursive subroutine cgetrf2(M, N, A, LDA, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void cgetrf2_(const armpl_int_t *m, const armpl_int_t *n,
              armpl_singlecomplex_t *a, const armpl_int_t *lda,
              armpl_int_t *ipiv, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the M-by-N matrix to be factored. On exit, the factors L and U from the factorization A = P*L*U; the unit diagonal elements of L are not stored.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

IPIV Output parameter.

IPIV is INTEGER array, dimension (min(M, N))

The pivot indices; for 1 <= i <= min(M, N), row i of the matrix was interchanged with row IPIV(i).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *dgetrf2*, *sgetrf2* and *zgetrf2*. It also exists with a native C interface as *LAPACKE_cgetrf2*.

4.3.18 cggqrf

cggqrf computes a generalized QR factorization of an N-by-M matrix A and an N-by-P matrix B:

```
A = Q*R,        B = Q*T*Z,
```

where Q is an N-by-N unitary matrix, Z is a P-by-P unitary matrix, and R and T assume one of the forms:

if N >= M, R = (R11) M , or if N < M, R = (R11 R12) N,

```
(  0  ) N-M                          N   M-N
   M
```

where R11 is upper triangular, and

if N <= P, T = (0 T12) N, or if N > P, T = (T11) N-P,

```
P-N  N                             ( T21 ) P
                                     P
```

where T12 or T21 is upper triangular.

In particular, if B is square and nonsingular, the GQR factorization of A and B implicitly gives the QR factorization of inv(B)*A:

```
inv(B)*A = Z**H * (inv(T)*R)
```

where inv(B) denotes the inverse of the matrix B, and Z' denotes the conjugate transpose of matrix Z.

Syntax

Fortran specification:

```
use armpl_library

subroutine cggqrf(N, M, P, A, LDA, TAUA, B, LDB, TAUB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cggqrf_(const armpl_int_t *n, const armpl_int_t *m, const armpl_int_t *p,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             armpl_singlecomplex_t *taua, armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, armpl_singlecomplex_t *taub,
             armpl_singlecomplex_t *work, const armpl_int_t *lwork,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The number of rows of the matrices A and B. N >= 0.

M Input parameter.

M is INTEGER

The number of columns of the matrix A. M >= 0.

P Input parameter.

P is INTEGER

The number of columns of the matrix B. P >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, M). On entry, the N-by-M matrix A. On exit, the elements on and above the diagonal of the array contain the min(N, M)-by-M upper trapezoidal matrix R (R is upper triangular if N >= M); the elements below the diagonal, with the array TAUA, represent the unitary matrix Q as a product of min(N, M) elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TAUA Output parameter.

TAUA is COMPLEX

TAUA is an array, dimension (min(N, M)). The scalar factors of the elementary reflectors which represent the unitary matrix Q (see Further Details).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, P). On entry, the N-by-P matrix B. On exit, if N <= P, the upper triangle of the subarray B(1:N,P-N+1:P) contains the N-by-N upper triangular matrix T; if N > P, the elements on and above the (N-P)-th subdiagonal contain the N-by-P upper trapezoidal matrix T; the remaining elements, with the array TAUB, represent the unitary matrix Z as a product of elementary reflectors (see Further Details).

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

TAUB Output parameter.

TAUB is COMPLEX

TAUB is an array, dimension (min(N, P)). The scalar factors of the elementary reflectors which represent the unitary matrix Z (see Further Details).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N, M, P). For optimum performance LWORK >= max(N, M, P)*max(NB1,NB2,NB3), where NB1 is the optimal blocksize for the QR factorization of an N-by-M matrix, NB2 is the optimal blocksize for the RQ factorization of an N-by-P matrix, and NB3 is the optimal blocksize for a call of CUNMQR.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *dggqrf*, *sggqrf* and *zggqrf*. It also exists with a native C interface as *LAPACKE_cggqrf*.

4.3.19 cggrqf

cggrqf computes a generalized RQ factorization of an M-by-N matrix A and a P-by-N matrix B:

```
A = R*Q,        B = Z*T*Q,
```

where Q is an N-by-N unitary matrix, Z is a P-by-P unitary matrix, and R and T assume one of the forms:

if M <= N, R = (0 R12) M, or if M > N, R = (R11) M-N,

```
N-M  M                            ( R21 ) N
                                    N
```

where R12 or R21 is upper triangular, and

if P >= N, T = (T11) N , or if P < N, T = (T11 T12) P,

```
(  0  ) P-N                         P   N-P
   N
```

where T11 is upper triangular.

In particular, if B is square and nonsingular, the GRQ factorization of A and B implicitly gives the RQ factorization of A*inv(B):

```
A*inv(B) = (R*inv(T))*Z**H
```

where inv(B) denotes the inverse of the matrix B, and Z^H denotes the conjugate transpose of the matrix Z.

Syntax

Fortran specification:

```
use armpl_library

subroutine cggrqf(M, P, N, A, LDA, TAUA, B, LDB, TAUB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cggrqf_(const armpl_int_t *m, const armpl_int_t *p, const armpl_int_t *n,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             armpl_singlecomplex_t *taua, armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, armpl_singlecomplex_t *taub,
             armpl_singlecomplex_t *work, const armpl_int_t *lwork,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

P Input parameter.

P is INTEGER

The number of rows of the matrix B. P >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrices A and B. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, if M <= N, the upper triangle of the subarray A(1:M,N-M+1:N) contains the M-by-M upper triangular matrix R; if M > N, the elements on and above the (M-N)-th subdiagonal contain the M-by-N upper trapezoidal matrix R; the remaining elements, with the array TAUA, represent the unitary matrix Q as a product of elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAUA Output parameter.

TAUA is COMPLEX

TAUA is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors which represent the unitary matrix Q (see Further Details).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, N). On entry, the P-by-N matrix B. On exit, the elements on and above the diagonal of the array contain the min(P, N)-by-N upper trapezoidal matrix T (T is upper triangular if P >= N); the elements below the diagonal, with the array TAUB, represent the unitary matrix Z as a product of elementary reflectors (see Further Details).

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, P).

TAUB Output parameter.

TAUB is COMPLEX

TAUB is an array, dimension (min(P, N)). The scalar factors of the elementary reflectors which represent the unitary matrix Z (see Further Details).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N, M, P). For optimum performance LWORK >= max(N, M, P)*max(NB1,NB2,NB3), where NB1 is the optimal blocksize for the RQ factorization of an M-by-N matrix, NB2 is the optimal blocksize for the QR factorization of a P-by-N matrix, and NB3 is the optimal blocksize for a call of CUNMRQ.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO=-i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *dggrqf*, *sggrqf* and *zggrqf*. It also exists with a native C interface as *LAPACKE_cggrqf*.

4.3.20 cgttrf

cgttrf computes an LU factorization of a complex tridiagonal matrix A using elimination with partial pivoting and row interchanges.

The factorization has the form

```
A = L * U
```

where L is a product of permutation and unit lower bidiagonal matrices and U is upper triangular with nonzeros in only the main diagonal and first two superdiagonals.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgttrf(N, DL, D, DU, DU2, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void cgttrf_(const armpl_int_t *n, armpl_singlecomplex_t *dl,
             armpl_singlecomplex_t *d, armpl_singlecomplex_t *du,
             armpl_singlecomplex_t *du2, armpl_int_t *ipiv,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A.

DL Input and output parameter.

DL is COMPLEX

DL is an array, dimension (N-1). On entry, DL must contain the (n-1) sub-diagonal elements of A.

On exit, DL is overwritten by the (n-1) multipliers that define the matrix L from the LU factorization of A.

D Input and output parameter.

D is COMPLEX

D is an array, dimension (N). On entry, D must contain the diagonal elements of A.

On exit, D is overwritten by the n diagonal elements of the upper triangular matrix U from the LU factorization of A.

DU Input and output parameter.

DU is COMPLEX

DU is an array, dimension (N-1). On entry, DU must contain the (n-1) super-diagonal elements of A.

On exit, DU is overwritten by the (n-1) elements of the first super-diagonal of U.

DU2 Output parameter.

DU2 is COMPLEX

DU2 is an array, dimension (N-2). On exit, DU2 is overwritten by the (n-2) elements of the second super-diagonal of U.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices; for 1 <= i <= n, row i of the matrix was interchanged with row IPIV(i). IPIV(i) will always be either i or i+1; IPIV(i) = i indicates a row interchange was not required.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value > 0: if INFO = k, U(k,k) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *dgttrf*, *sgttrf* and *zgttrf*. It also exists with a native C interface as *LAPACKE_cgttrf*.

4.3.21 chetrf

chetrf computes the factorization of a complex Hermitian matrix A using the Bunch-Kaufman diagonal pivoting method. The form of the factorization is

```
A = U*D*U**H  or  A = L*D*L**H
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is Hermitian and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine chetrf(UPLO, N, A, LDA, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void chetrf_(const char *uplo, const armpl_int_t *n, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *ipiv,
             armpl_singlecomplex_t *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L (see below for further details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >=1. For best performance LWORK >= N*NB, where NB is the block size returned by ILAENV.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *zhetrf*. It also exists with a native C interface as *LAPACKE_chetrf*.

4.3.22 chetrf_aa

CHETRF_AA computes the factorization of a complex hermitian matrix A using the Aasen's algorithm. The form of the factorization is

```
A = U*T*U**H  or  A = L*T*L**H
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and T is a hermitian tridiagonal matrix.

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine chetrf_aa(UPLO, N, A, LDA, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void chetrf_aa_(const char *uplo, const armpl_int_t *n,
                armpl_singlecomplex_t *a, const armpl_int_t *lda,
                armpl_int_t *ipiv, armpl_singlecomplex_t *work,
                const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, the tridiagonal matrix is stored in the diagonals and the subdiagonals of A just below (or above) the diagonals, and L is stored below (or above) the subdiaonals, when UPLO is 'L' (or 'U').

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

On exit, it contains the details of the interchanges, i.e., the row and column k of A were interchanged with the row and column IPIV(k).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= 2*N. For optimum performance LWORK >= N*(1+NB), where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *zhetrf_aa*. It also exists with a native C interface as *LAPACKE_chetrf_aa*.

4.3.23 chetrf_rk

CHETRF_RK computes the factorization of a complex Hermitian matrix A using the bounded Bunch-Kaufman (rook) diagonal pivoting method:

```
A = P*U*D*(U**H)*(P**T) or A = P*L*D*(L**H)*(P**T),
```

where U (or L) is unit upper (or lower) triangular matrix, U^H (or L^H) is the conjugate of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is Hermitian and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the blocked version of the algorithm, calling Level 3 BLAS. For more information see Further Details section.

Syntax

Fortran specification:

```
use armpl_library

subroutine chetrf_rk(UPLO, N, A, LDA, E, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void chetrf_rk_(const char *uplo, const armpl_int_t *n,
                armpl_singlecomplex_t *a, const armpl_int_t *lda,
                armpl_singlecomplex_t *e, armpl_int_t *ipiv,
                armpl_singlecomplex_t *work, const armpl_int_t *lwork,
                armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the Hermitian matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U': the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced.

If UPLO = 'L': the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, contains: a) ONLY diagonal elements of the Hermitian block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Output parameter.

E is COMPLEX

E is an array, dimension (N). On exit, contains the superdiagonal (or subdiagonal) elements of the Hermitian block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) is set to 0; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) is set to 0.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is set to 0 in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

IPIV describes the permutation matrix P in the factorization of matrix A as follows. The absolute value of IPIV(k) represents the index of row and column that were interchanged with the k-th row and column. The value of UPLO describes the order in which the interchanges were applied. Also, the sign of IPIV represents the block structure of the Hermitian block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks which correspond to 1 or 2 interchanges at each factorization step. For more info see Further Details section.

If UPLO = 'U', (in factorization order, k decreases from N to 1): a) A single positive entry IPIV(k) > 0 means: D(k,k) is a 1-by-1 diagonal block. If IPIV(k) != k, rows and columns k and IPIV(k) were interchanged in the matrix A(1:N,1:N); If IPIV(k) = k, no interchange occurred.

b) A pair of consecutive negative entries IPIV(k) < 0 and IPIV(k-1) < 0 means: D(k-1:k,k-1:k) is a 2-by-2 diagonal block. (NOTE: negative entries in IPIV appear ONLY in pairs). 1) If -IPIV(k) != k, rows and columns k and -IPIV(k) were interchanged in the matrix A(1:N,1:N). If -IPIV(k) = k, no interchange occurred. 2) If -IPIV(k-1) != k-1, rows and columns k-1 and -IPIV(k-1) were interchanged in the matrix A(1:N,1:N). If -IPIV(k-1) = k-1, no interchange occurred.

c) In both cases a) and b), always ABS(IPIV(k)) <= k.

d) NOTE: Any entry IPIV(k) is always NONZERO on output.

If UPLO = 'L', (in factorization order, k increases from 1 to N): a) A single positive entry IPIV(k) > 0 means: D(k,k) is a 1-by-1 diagonal block. If IPIV(k) != k, rows and columns k and IPIV(k) were interchanged in the matrix A(1:N,1:N). If IPIV(k) = k, no interchange occurred.

b) A pair of consecutive negative entries IPIV(k) < 0 and IPIV(k+1) < 0 means: D(k:k+1,k:k+1) is a 2-by-2 diagonal block. (NOTE: negative entries in IPIV appear ONLY in pairs). 1) If -IPIV(k) != k, rows and columns k and -IPIV(k) were interchanged in the matrix A(1:N,1:N). If -IPIV(k) = k, no interchange occurred. 2) If -IPIV(k+1) != k+1, rows and columns k-1 and -IPIV(k-1) were interchanged in the matrix A(1:N,1:N). If -IPIV(k+1) = k+1, no interchange occurred.

c) In both cases a) and b), always ABS(IPIV(k)) >= k.

d) NOTE: Any entry IPIV(k) is always NONZERO on output.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)).. On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >=1. For best performance LWORK >= N*NB, where NB is the block size returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit

< 0: If INFO = -k, the k-th argument had an illegal value

> 0: If INFO = k, the matrix A is singular, because: If UPLO = 'U': column k in the upper triangular part of A contains all zeros. If UPLO = 'L': column k in the lower triangular part of A contains all zeros.

Therefore D(k,k) is exactly zero, and superdiagonal elements of column k of U (or subdiagonal elements of column k of L) are all zeros. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

NOTE: INFO only stores the first occurrence of a singularity, any subsequent occurrence of singularity is not stored in INFO even though the factorization always completes.

Related Information

For this routine in other precisions, please see *zhetrf_rk*. It also exists with a native C interface as *LAPACKE_chetrf_rk*.

4.3.24 chetrf_rook

CHETRF_ROOK computes the factorization of a comlex Hermitian matrix A using the bounded Bunch-Kaufman ("rook") diagonal pivoting method. The form of the factorization is

```
A = U*D*U**T  or  A = L*D*L**T
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is Hermitian and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine chetrf_rook(UPLO, N, A, LDA, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void chetrf_rook_(const char *uplo, const armpl_int_t *n,
                  armpl_singlecomplex_t *a, const armpl_int_t *lda,
                  armpl_int_t *ipiv, armpl_singlecomplex_t *work,
                  const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L (see below for further details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D.

If UPLO = 'U': Only the last KB elements of IPIV are set.

If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k-1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k-1 and -IPIV(k-1) were inerchaged, D(k-1:k,k-1:k) is a 2-by-2 diagonal block.

If UPLO = 'L': Only the first KB elements of IPIV are set.

If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k+1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k+1 and -IPIV(k+1) were inerchaged, D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)).. On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >=1. For best performance LWORK >= N*NB, where NB is the block size returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *zhetrf_rook*. It also exists with a native C interface as *LAPACKE_chetrf_rook*.

4.3.25 chptrf

chptrf computes the factorization of a complex Hermitian packed matrix A using the Bunch-Kaufman diagonal pivoting method:

```
A = U*D*U**H  or  A = L*D*L**H
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is Hermitian and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

Syntax

Fortran specification:

```
use armpl_library

subroutine chptrf(UPLO, N, AP, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void chptrf_(const char *uplo, const armpl_int_t *n,
             armpl_singlecomplex_t *ap, armpl_int_t *ipiv, armpl_int_t *info,
             ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the Hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L, stored as a packed triangular matrix overwriting A (see below for further details).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *zhptrf*. It also exists with a native C interface as *LAPACKE_chptrf*.

4.3.26 cpbtrf

cpbtrf computes the Cholesky factorization of a complex Hermitian positive definite band matrix A.

The factorization has the form

```
A = U**H * U,  if UPLO = 'U', or
A = L  * L**H,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is lower triangular.

Syntax

Fortran specification:

```
use armpl_library

subroutine cpbtrf(UPLO, N, KD, AB, LDAB, INFO)
```

C specification:

```
#include "armpl.h"

void cpbtrf_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
             armpl_singlecomplex_t *ab, const armpl_int_t *ldab,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the Hermitian band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, if INFO = 0, the triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H of the band matrix A, in the same storage format as A.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i is not positive definite, and the factorization could not be completed.

Related Information

For this routine in other precisions, please see *dpbtrf*, *spbtrf* and *zpbtrf*. It also exists with a native C interface as *LAPACKE_cpbtrf*.

4.3.27 cpftrf

cpftrf computes the Cholesky factorization of a complex Hermitian positive definite matrix A.

The factorization has the form

```
A = U**H * U,  if UPLO = 'U', or
A = L  * L**H,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is lower triangular.

This is the block version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine cpftrf(TRANSR, UPLO, N, A, INFO)
```

C specification:

```
#include "armpl.h"

void cpftrf_(const char *transr, const char *uplo, const armpl_int_t *n,
             armpl_singlecomplex_t *a, armpl_int_t *info, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': The Normal TRANSR of RFP A is stored; = 'C': The Conjugate-transpose TRANSR of RFP A is stored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of RFP A is stored; = 'L': Lower triangle of RFP A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (N*(N+1)/2);. On entry, the Hermitian matrix A in RFP format. RFP format is described by TRANSR, UPLO, and N as follows: If TRANSR = 'N' then RFP A is (0:N,0:k-1) when N is even; k=N/2. RFP A is (0:N-1,0:k) when N is odd; k=N/2. IF TRANSR = 'C' then RFP is the Conjugate-transpose of RFP A as defined when TRANSR = 'N'. The contents of RFP A are defined by UPLO as follows: If UPLO = 'U' the RFP A contains the nt elements of upper packed A. If UPLO = 'L' the RFP A contains the elements of lower packed A. The LDA of RFP A is (N+1)/2 when TRANSR = 'C'. When TRANSR is 'N' the LDA is N+1 when N is even and N is odd. See the Note below for more details.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization RFP A = U^H *U or RFP A = L*L^H .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i is not positive definite, and the factorization could not be completed.

Further Notes on RFP Format:

We first consider Standard Packed Format when N is even. We give an example where N = 6.

AP is Upper AP is Lower

00 01 02 03 04 05 00 11 12 13 14 15 10 11 22 23 24 25 20 21 22 33 34 35 30 31 32 33 44 45 40 41 42 43 44 55 50 51 52 53 54 55

Let TRANSR = 'N'. RFP holds AP as follows: For UPLO = 'U' the upper trapezoid A(0:5,0:2) consists of the last three columns of AP upper. The lower triangle A(4:6,0:2) consists of conjugate-transpose of the first three columns of AP upper. For UPLO = 'L' the lower trapezoid A(1:6,0:2) consists of the first three columns of AP lower. The upper triangle A(0:2,0:2) consists of conjugate-transpose of the last three columns of AP lower. To denote conjugate we place – above the element. This covers the case N even and TRANSR = 'N'.

RFP A RFP A

– – – 03 04 05 33 43 53 – – 13 14 15 00 44 54 – 23 24 25 10 11 55

33 34 35 20 21 22 – 00 44 45 30 31 32 – – 01 11 55 40 41 42 – – – 02 12 22 50 51 52

Now let TRANSR = 'C'. RFP A in both UPLO cases is just the conjugate- transpose of RFP A above. One therefore gets:

RFP A RFP A

– – – – – – – – – – 03 13 23 33 00 01 02 33 00 10 20 30 40 50 – – – – – – – – – – 04 14 24 34 44 11 12 43 44 11 21 31 41 51 – – – – – – – – – – 05 15 25 35 45 55 22 53 54 55 22 32 42 52

We next consider Standard Packed Format when N is odd. We give an example where N = 5.

AP is Upper AP is Lower

00 01 02 03 04 00 11 12 13 14 10 11 22 23 24 20 21 22 33 34 30 31 32 33 44 40 41 42 43 44

Let TRANSR = 'N'. RFP holds AP as follows: For UPLO = 'U' the upper trapezoid A(0:4,0:2) consists of the last three columns of AP upper. The lower triangle A(3:4,0:1) consists of conjugate-transpose of the first two columns of AP upper. For UPLO = 'L' the lower trapezoid A(0:4,0:2) consists of the first three columns of AP lower. The upper triangle A(0:1,1:2) consists of conjugate-transpose of the last two columns of AP lower. To denote conjugate we place – above the element. This covers the case N odd and TRANSR = 'N'.

RFP A RFP A

– – 02 03 04 00 33 43 – 12 13 14 10 11 44

22 23 24 20 21 22 – 00 33 34 30 31 32 – – 01 11 44 40 41 42

Now let TRANSR = 'C'. RFP A in both UPLO cases is just the conjugate- transpose of RFP A above. One therefore gets:

RFP A RFP A

– – – – – – – – – 02 12 22 00 01 00 10 20 30 40 50 – – – – – – – – – 03 13 23 33 11 33 11 21 31 41 51 – – – – – – – – – – 04 14 24 34 44 43 44 22 32 42 52

Related Information

For this routine in other precisions, please see *dpftrf*, *spftrf* and *zpftrf*. It also exists with a native C interface as *LAPACKE_cpftrf*.

4.3.28 cpotrf

cpotrf computes the Cholesky factorization of a complex Hermitian positive definite matrix A.

The factorization has the form

```
A = U**H * U,  if UPLO = 'U', or
A = L  * L**H,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is lower triangular.

This is the block version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine cpotrf(UPLO, N, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void cpotrf_(const char *uplo, const armpl_int_t *n, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H .

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i is not positive definite, and the factorization could not be completed.

Related Information

For this routine in other precisions, please see *dpotrf*, *spotrf* and *zpotrf*. It also exists with a native C interface as *LAPACKE_cpotrf*.

4.3.29 cpotrf2

`cpotrf2` computes the Cholesky factorization of a real symmetric positive definite matrix A using the recursive algorithm.

The factorization has the form

```
A = U**H * U,  if UPLO = 'U', or
A = L  * L**H,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is lower triangular.

This is the recursive version of the algorithm. It divides the matrix into four submatrices:

```
    [  A11 | A12  ]  where A11 is n1 by n1 and A22 is n2 by n2
A = [ -----|-----  ]  with n1 = n/2
    [  A21 | A22  ]       n2 = n-n1
```

The subroutine calls itself to factor A11. Update and scale A21 or A12, update A22 then calls itself to factor A22.

Syntax

Fortran specification:

```
use armpl_library

recursive subroutine cpotrf2(UPLO, N, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void cpotrf2_(const char *uplo, const armpl_int_t *n,
              armpl_singlecomplex_t *a, const armpl_int_t *lda,
              armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

= '`U`': Upper triangle of `A` is stored; = '`L`': Lower triangle of `A` is stored.

N Input parameter.

`N` is `INTEGER`

The order of the matrix `A`. `N` >= 0.

A Input and output parameter.

`A` is `COMPLEX`

`A` is an array, dimension (`LDA`, `N`). On entry, the symmetric matrix `A`. If `UPLO` = '`U`', the leading N-by-N upper triangular part of `A` contains the upper triangular part of the matrix `A`, and the strictly lower triangular part of `A` is not referenced. If `UPLO` = '`L`', the leading N-by-N lower triangular part of `A` contains the lower triangular part of the matrix `A`, and the strictly upper triangular part of `A` is not referenced.

On exit, if `INFO` = 0, the factor U or L from the Cholesky factorization `A` = U^H *U or `A` = L*L^H .

LDA Input parameter.

`LDA` is `INTEGER`

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i is not positive definite, and the factorization could not be completed.

Related Information

For this routine in other precisions, please see *dpotrf2*, *spotrf2* and *zpotrf2*. It also exists with a native C interface as *LAPACKE_cpotrf2*.

4.3.30 cpptrf

cpptrf computes the Cholesky factorization of a complex Hermitian positive definite matrix A stored in packed format.

The factorization has the form

```
A = U**H * U,  if UPLO = 'U', or
A = L  * L**H,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is lower triangular.

Syntax

Fortran specification:

```
use armpl_library

subroutine cpptrf(UPLO, N, AP, INFO)
```

C specification:

```
#include "armpl.h"

void cpptrf_(const char *uplo, const armpl_int_t *n,
             armpl_singlecomplex_t *ap, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the Hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO =

'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n. See below for further details.

On exit, if INFO = 0, the triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H , in the same storage format as A.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i is not positive definite, and the factorization could not be completed.

Related Information

For this routine in other precisions, please see *dpptrf*, *spptrf* and *zpptrf*. It also exists with a native C interface as *LAPACKE_cpptrf*.

4.3.31 cpstrf

cpstrf computes the Cholesky factorization with complete pivoting of a complex Hermitian positive semidefinite matrix A.

The factorization has the form

```
P**T * A * P = U**H * U ,  if UPLO = 'U',
P**T * A * P = L  * L**H,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is lower triangular, and P is stored as vector PIV.

This algorithm does not attempt to check that A is positive semidefinite. This version of the algorithm calls level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine cpstrf(UPLO, N, A, LDA, PIV, RANK, TOL, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void cpstrf_(const char *uplo, const armpl_int_t *n, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *piv, armpl_int_t *rank,
             const float *tol, float *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored. = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading n by n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n by n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization as above.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

PIV Output parameter.

PIV is INTEGER array, dimension (N)

PIV is such that the nonzero entries are P(PIV(K), K) = 1.

RANK Output parameter.

RANK is INTEGER

The rank of A given by the number of steps the algorithm completed.

TOL Input parameter.

TOL is REAL

User defined tolerance. If TOL < 0, then N*U*MAX(A(K,K)) will be used. The algorithm terminates at the (K-1)st step if the pivot <= TOL.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (2*N). Work space.

INFO Output parameter.

INFO is INTEGER

< 0: If INFO = -K, the K-th argument had an illegal value, = 0: algorithm completed successfully, and > 0: the matrix A is either rank deficient with computed rank as returned in RANK, or is not positive semidefinite. See Section 7 of LAPACK Working Note #161 for further information.

Related Information

For this routine in other precisions, please see *dpstrf*, *spstrf* and *zpstrf*. It also exists with a native C interface as *LAPACKE_cpstrf*.

4.3.32 cpttrf

cpttrf computes the L*D*L^H factorization of a complex Hermitian positive definite tridiagonal matrix A. The factorization may also be regarded as having the form A = U^H *D*U.

Syntax

Fortran specification:

```
use armpl_library

subroutine cpttrf(N, D, E, INFO)
```

C specification:

```
#include "armpl.h"

void cpttrf_(const armpl_int_t *n, float *d, armpl_singlecomplex_t *e,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry, the n diagonal elements of the tridiagonal matrix A. On exit, the n diagonal elements of the diagonal matrix D from the L*D*L^H factorization of A.

E Input and output parameter.

E is COMPLEX

E is an array, dimension (N-1). On entry, the (n-1) subdiagonal elements of the tridiagonal matrix A. On exit, the (n-1) subdiagonal elements of the unit bidiagonal factor L from the L*D*L^H factorization of A. E can also be regarded as the superdiagonal of the unit bidiagonal factor U from the U^H *D*U factorization of A.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value > 0: if INFO = k, the leading minor of order k is not positive definite; if k < N, the factorization could not be completed, while if k = N, the factorization was completed, but D(N) <= 0.

Related Information

For this routine in other precisions, please see *dpttrf*, *spttrf* and *zpttrf*. It also exists with a native C interface as *LAPACKE_cpttrf*.

4.3.33 csptrf

csptrf computes the factorization of a complex symmetric matrix A stored in packed format using the Bunch-Kaufman diagonal pivoting method:

```
A = U*D*U**T  or  A = L*D*L**T
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

Syntax

Fortran specification:

```
use armpl_library

subroutine csptrf(UPLO, N, AP, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void csptrf_(const char *uplo, const armpl_int_t *n,
             armpl_singlecomplex_t *ap, armpl_int_t *ipiv, armpl_int_t *info,
             ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L, stored as a packed triangular matrix overwriting A (see below for further details).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *dsptrf*, *ssptrf* and *zsptrf*. It also exists with a native C interface as *LAPACKE_csptrf*.

4.3.34 csytrf

csytrf computes the factorization of a complex symmetric matrix A using the Bunch-Kaufman diagonal pivoting method. The form of the factorization is

```
A = U*D*U**T  or  A = L*D*L**T
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is symmetric and block diagonal with with 1-by-1 and 2-by-2 diagonal blocks.

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine csytrf(UPLO, N, A, LDA, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void csytrf_(const char *uplo, const armpl_int_t *n, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *ipiv,
             armpl_singlecomplex_t *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L (see below for further details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >=1. For best performance LWORK >= N*NB, where NB is the block size returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *dsytrf*, *ssytrf* and *zsytrf*. It also exists with a native C interface as *LAPACKE_csytrf*.

4.3.35 csytrf_aa

CSYTRF_AA computes the factorization of a complex symmetric matrix A using the Aasen's algorithm. The form of the factorization is

```
A = U*T*U**T  or  A = L*T*L**T
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and T is a complex symmetric tridiagonal matrix.

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine csytrf_aa(UPLO, N, A, LDA, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void csytrf_aa_(const char *uplo, const armpl_int_t *n,
                armpl_singlecomplex_t *a, const armpl_int_t *lda,
                armpl_int_t *ipiv, armpl_singlecomplex_t *work,
                const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, the tridiagonal matrix is stored in the diagonals and the subdiagonals of A just below (or above) the diagonals, and L is stored below (or above) the subdiaonals, when UPLO is 'L' (or 'U').

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

On exit, it contains the details of the interchanges, i.e., the row and column k of A were interchanged with the row and column IPIV(k).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= MAX(1,2*N). For optimum performance LWORK >= N*(1+NB), where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *dsytrf_aa*, *ssytrf_aa* and *zsytrf_aa*. It also exists with a native C interface as *LAPACKE_csytrf_aa*.

4.3.36 csytrf_rk

CSYTRF_RK computes the factorization of a complex symmetric matrix A using the bounded Bunch-Kaufman (rook) diagonal pivoting method:

```
A = P*U*D*(U**T)*(P**T) or A = P*L*D*(L**T)*(P**T),
```

where U (or L) is unit upper (or lower) triangular matrix, U^T (or L^T) is the transpose of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the blocked version of the algorithm, calling Level 3 BLAS. For more information see Further Details section.

Syntax

Fortran specification:

```
use armpl_library

subroutine csytrf_rk(UPLO, N, A, LDA, E, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void csytrf_rk_(const char *uplo, const armpl_int_t *n,
                armpl_singlecomplex_t *a, const armpl_int_t *lda,
                armpl_singlecomplex_t *e, armpl_int_t *ipiv,
                armpl_singlecomplex_t *work, const armpl_int_t *lwork,
                armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U': the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced.

If UPLO = 'L': the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, contains: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Output parameter.

E is COMPLEX

E is an array, dimension (N). On exit, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) is set to 0; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) is set to 0.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is set to 0 in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

IPIV describes the permutation matrix P in the factorization of matrix A as follows. The absolute value of IPIV(k) represents the index of row and column that were interchanged with the k-th row and column. The value of UPLO describes the order in which the interchanges were applied. Also, the sign of IPIV represents the block structure of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks which correspond to 1 or 2 interchanges at each factorization step. For more info see Further Details section.

If UPLO = 'U', (in factorization order, k decreases from N to 1): a) A single positive entry IPIV(k) > 0 means: D(k,k) is a 1-by-1 diagonal block. If IPIV(k) != k, rows and columns k and IPIV(k) were interchanged in the matrix A(1:N,1:N); If IPIV(k) = k, no interchange occurred.

b) A pair of consecutive negative entries IPIV(k) < 0 and IPIV(k-1) < 0 means: D(k-1:k,k-1:k) is a 2-by-2 diagonal block. (NOTE: negative entries in IPIV appear ONLY in pairs). 1) If -IPIV(k) != k, rows and columns k and -IPIV(k) were interchanged in the matrix A(1:N,1:N). If -IPIV(k) = k, no interchange occurred. 2) If -IPIV(k-1) != k-1, rows and columns k-1 and -IPIV(k-1) were interchanged in the matrix A(1:N,1:N). If -IPIV(k-1) = k-1, no interchange occurred.

c) In both cases a) and b), always ABS(IPIV(k)) <= k.

d) NOTE: Any entry IPIV(k) is always NONZERO on output.

If UPLO = 'L', (in factorization order, k increases from 1 to N): a) A single positive entry IPIV(k) > 0 means: D(k,k) is a 1-by-1 diagonal block. If IPIV(k) != k, rows and columns k and IPIV(k) were interchanged in the matrix A(1:N,1:N). If IPIV(k) = k, no interchange occurred.

b) A pair of consecutive negative entries IPIV(k) < 0 and IPIV(k+1) < 0 means: D(k:k+1,k:k+1) is a 2-by-2 diagonal block. (NOTE: negative entries in IPIV appear ONLY in pairs). 1) If -IPIV(k) != k, rows and columns k and -IPIV(k) were interchanged in the matrix A(1:N,1:N). If -IPIV(k) = k, no interchange occurred. 2) If -IPIV(k+1) != k+1, rows and columns k-1 and -IPIV(k-1) were interchanged in the matrix A(1:N,1:N). If -IPIV(k+1) = k+1, no interchange occurred.

c) In both cases a) and b), always ABS(IPIV(k)) >= k.

d) NOTE: Any entry IPIV(k) is always NONZERO on output.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)).. On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >=1. For best performance LWORK >= N*NB, where NB is the block size returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit

< 0: If INFO = -k, the k-th argument had an illegal value

> 0: If INFO = k, the matrix A is singular, because: If UPLO = 'U': column k in the upper triangular part of A contains all zeros. If UPLO = 'L': column k in the lower triangular part of A contains all zeros.

Therefore D(k,k) is exactly zero, and superdiagonal elements of column k of U (or subdiagonal elements of column k of L) are all zeros. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

NOTE: INFO only stores the first occurrence of a singularity, any subsequent occurrence of singularity is not stored in INFO even though the factorization always completes.

Related Information

For this routine in other precisions, please see *dsytrf_rk*, *ssytrf_rk* and *zsytrf_rk*. It also exists with a native C interface as *LAPACKE_csytrf_rk*.

4.3.37 csytrf_rook

CSYTRF_ROOK computes the factorization of a complex symmetric matrix A using the bounded Bunch-Kaufman ("rook") diagonal pivoting method. The form of the factorization is

```
A = U*D*U**T  or  A = L*D*L**T
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine csytrf_rook(UPLO, N, A, LDA, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void csytrf_rook_(const char *uplo, const armpl_int_t *n,
                  armpl_singlecomplex_t *a, const armpl_int_t *lda,
                  armpl_int_t *ipiv, armpl_singlecomplex_t *work,
                  const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L (see below for further details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D.

If UPLO = 'U': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k-1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k-1 and -IPIV(k-1) were inerchaged, D(k-1:k,k-1:k) is a 2-by-2 diagonal block.

If UPLO = 'L': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k+1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k+1 and -IPIV(k+1) were inerchaged, D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)).. On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >=1. For best performance LWORK >= N*NB, where NB is the block size returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *dsytrf_rook*, *ssytrf_rook* and *zsytrf_rook*. It also exists with a native C interface as *LAPACKE_csytrf_rook*.

4.3.38 ctplqt

ctplqt computes a blocked LQ factorization of a complex "triangular-pentagonal" matrix C, which is composed of a triangular block A and pentagonal block B, using the compact WY representation for Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine ctplqt(M, N, L, MB, A, LDA, B, LDB, T, LDT, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void ctplqt_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *l,
             const armpl_int_t *mb, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, armpl_singlecomplex_t *t,
             const armpl_int_t *ldt, armpl_singlecomplex_t *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix B, and the order of the triangular matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix B. N >= 0.

L Input parameter.

L is INTEGER

The number of rows of the lower trapezoidal part of B. MIN(M, N) >= L >= 0. See Further Details.

MB Input parameter.

MB is INTEGER

The block size to be used in the blocked QR. M >= MB >= 1.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, M). On entry, the lower triangular M-by-M matrix A. On exit, the elements on and below the diagonal of the array contain the lower triangular matrix L.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, N). On entry, the pentagonal M-by-N matrix B. The first N-L columns are rectangular, and the last L columns are lower trapezoidal. On exit, B contains the pentagonal matrix V. See Further Details.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M).

T Output parameter.

T is COMPLEX

T is an array, dimension (LDT, N). The lower triangular block reflectors stored in compact form as a sequence of upper triangular blocks. See Further Details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= MB.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MB*M) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dtplqt*, *stplqt* and *ztplqt*.

4.3.39 ctpmlqt

ctpmlqt applies a complex orthogonal matrix Q obtained from a "triangular-pentagonal" complex block reflector H to a general complex matrix C, which consists of two blocks A and B.

Syntax

Fortran specification:

```
use armpl_library

subroutine ctpmlqt(SIDE, TRANS, M, N, K, L, MB, V, LDV, T, LDT, A, LDA, B,
                   LDB, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void ctpmlqt_(const char *side, const char *trans, const armpl_int_t *m,
              const armpl_int_t *n, const armpl_int_t *k,
              const armpl_int_t *l, const armpl_int_t *mb,
              const armpl_singlecomplex_t *v, const armpl_int_t *ldv,
              const armpl_singlecomplex_t *t, const armpl_int_t *ldt,
              armpl_singlecomplex_t *a, const armpl_int_t *lda,
              armpl_singlecomplex_t *b, const armpl_int_t *ldb,
              armpl_singlecomplex_t *work, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left; = 'R': apply Q or Q^H from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'C': Transpose, apply Q^H .

M Input parameter.

M is INTEGER

The number of rows of the matrix B. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix B. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q.

L Input parameter.

L is INTEGER

The order of the trapezoidal part of V. K >= L >= 0. See Further Details.

MB Input parameter.

MB is INTEGER

The block size used for the storage of T. K >= MB >= 1. This must be the same value of MB used to generate T in DTPLQT.

V Input parameter.

V is COMPLEX

V is an array, dimension (LDA, K). The i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,. . . ,k, as returned by DTPLQT in B. See Further Details.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. If SIDE = 'L', LDV >= max(1, M); if SIDE = 'R', LDV >= max(1, N).

T Input parameter.

T is COMPLEX

T is an array, dimension (LDT, K). The upper triangular factors of the block reflectors as returned by DTPLQT, stored as a MB-by-K matrix.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= MB.

A Input and output parameter.

A is COMPLEX

A is an array, dimension. (LDA, N) if SIDE = 'L' or (LDA, K) if SIDE = 'R' On entry, the K-by-N or M-by-K matrix A. On exit, A is overwritten by the corresponding block of Q*C or Q^H *C or C*Q or C*Q^H . See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDC >= max(1, K); If SIDE = 'R', LDC >= max(1, M).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, N). On entry, the M-by-N matrix B. On exit, B is overwritten by the corresponding block of Q*C or Q^H *C or C*Q or C*Q^H . See Further Details.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M).

WORK Output parameter.

WORK is COMPLEX array. The dimension of WORK is

N* MB if SIDE = 'L', or M* MB if SIDE = 'R'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dtpmlqt*, *stpmlqt* and *ztpmlqt*.

4.3.40 ctpmqrt

ctpmqrt applies a complex orthogonal matrix Q obtained from a "triangular-pentagonal" complex block reflector H to a general complex matrix C, which consists of two blocks A and B.

Syntax

Fortran specification:

```
use armpl_library

subroutine ctpmqrt(SIDE, TRANS, M, N, K, L, NB, V, LDV, T, LDT, A, LDA, B,
                   LDB, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void ctpmqrt_(const char *side, const char *trans, const armpl_int_t *m,
              const armpl_int_t *n, const armpl_int_t *k,
              const armpl_int_t *l, const armpl_int_t *nb,
              const armpl_singlecomplex_t *v, const armpl_int_t *ldv,
              const armpl_singlecomplex_t *t, const armpl_int_t *ldt,
              armpl_singlecomplex_t *a, const armpl_int_t *lda,
              armpl_singlecomplex_t *b, const armpl_int_t *ldb,
              armpl_singlecomplex_t *work, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left; = 'R': apply Q or Q^H from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'C': Transpose, apply Q^H .

M Input parameter.

M is INTEGER

The number of rows of the matrix B. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix B. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q.

L Input parameter.

L is INTEGER

The order of the trapezoidal part of V. K >= L >= 0. See Further Details.

NB Input parameter.

NB is INTEGER

The block size used for the storage of T. K >= NB >= 1. This must be the same value of NB used to generate T in CTPQRT.

V Input parameter.

V is COMPLEX

V is an array, dimension (LDA, K). The i-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by CTPQRT in B. See Further Details.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. If SIDE = 'L', LDV >= max(1, M); if SIDE = 'R', LDV >= max(1, N).

T Input parameter.

T is COMPLEX

T is an array, dimension (LDT, K). The upper triangular factors of the block reflectors as returned by CTPQRT, stored as a NB-by-K matrix.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= NB.

A Input and output parameter.

A is COMPLEX

A is an array, dimension. (LDA, N) if SIDE = 'L' or (LDA, K) if SIDE = 'R' On entry, the K-by-N or M-by-K matrix A. On exit, A is overwritten by the corresponding block of Q*C or Q^H *C or C*Q or C*Q^H . See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDC >= max(1, K); If SIDE = 'R', LDC >= max(1, M).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, N). On entry, the M-by-N matrix B. On exit, B is overwritten by the corresponding block of Q*C or Q^H *C or C*Q or C*Q^H . See Further Details.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M).

WORK Output parameter.

WORK is COMPLEX array. The dimension of WORK is

N* NB if SIDE = 'L', or M* NB if SIDE = 'R'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dtpmqrt*, *stpmqrt* and *ztpmqrt*. It also exists with a native C interface as *LAPACKE_ctpmqrt*.

4.3.41 ctpqrt

ctpqrt computes a blocked QR factorization of a complex "triangular-pentagonal" matrix C, which is composed of a triangular block A and pentagonal block B, using the compact WY representation for Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine ctpqrt(M, N, L, NB, A, LDA, B, LDB, T, LDT, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void ctpqrt_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *l,
             const armpl_int_t *nb, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, armpl_singlecomplex_t *t,
             const armpl_int_t *ldt, armpl_singlecomplex_t *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix B. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix B, and the order of the triangular matrix A. N >= 0.

L Input parameter.

L is INTEGER

The number of rows of the upper trapezoidal part of B. MIN(M, N) >= L >= 0. See Further Details.

NB Input parameter.

NB is INTEGER

The block size to be used in the blocked QR. N >= NB >= 1.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the upper triangular N-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the upper triangular matrix R.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, N). On entry, the pentagonal M-by-N matrix B. The first M-L rows are rectangular, and the last L rows are upper trapezoidal. On exit, B contains the pentagonal matrix V. See Further Details.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M).

T Output parameter.

T is COMPLEX

T is an array, dimension (LDT, N). The upper triangular block reflectors stored in compact form as a sequence of upper triangular blocks. See Further Details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= NB.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (NB*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dtpqrt*, *stpqrt* and *ztpqrt*. It also exists with a native C interface as *LAPACKE_ctpqrt*.

4.3.42 ctzrzf

ctzrzf reduces the M-by-N (M<=N) complex upper trapezoidal matrix A to upper triangular form by means of unitary transformations.

The upper trapezoidal matrix A is factored as

```
A = ( R  0 ) * Z,
```

where Z is an N-by-N unitary matrix and R is an M-by-M upper triangular matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine ctzrzf(M, N, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ctzrzf_(const armpl_int_t *m, const armpl_int_t *n,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             armpl_singlecomplex_t *tau, armpl_singlecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= M.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the leading M-by-N upper trapezoidal part of the array A must contain the matrix to be factorized. On exit, the leading M-by-M upper triangular part of A contains the upper triangular matrix R, and elements M+1 to N of the first M rows of A, with the array TAU, represent the unitary matrix Z as a product of M elementary reflectors.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is COMPLEX

TAU is an array, dimension (M). The scalar factors of the elementary reflectors.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, M). For optimum performance LWORK >= M*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dtzrzf*, *stzrzf* and *ztzrzf*. It also exists with a native C interface as *LAPACKE_ctzrzf*.

4.3.43 cunglq

`cunglq` generates an M-by-N complex matrix Q with orthonormal rows, which is defined as the first M rows of a product of K elementary reflectors of order N

```
Q  =  H(k)**H . . . H(2)**H H(1)**H
```

as returned by CGELQF.

Syntax

Fortran specification:

```
use armpl_library

subroutine cunglq(M, N, K, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cunglq_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *k,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const armpl_singlecomplex_t *tau, armpl_singlecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix Q. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix Q. N >= M.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. M >= K >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,. . . ,k, as returned by CGELQF in the first k rows of its array argument A. On exit, the M-by-N matrix Q.

LDA Input parameter.

LDA is INTEGER

The first dimension of the array A. LDA >= max(1, M).

TAU Input parameter.

TAU is COMPLEX

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by CGELQF.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, M). For optimum performance LWORK >= M*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit; < 0: if INFO = -i, the i-th argument has an illegal value

Related Information

For this routine in other precisions, please see *zunglq*. It also exists with a native C interface as *LAPACKE_cunglq*.

4.3.44 cungql

cungql generates an M-by-N complex matrix Q with orthonormal columns, which is defined as the last N columns of a product of K elementary reflectors of order M

```
Q  =  H(k) . . . H(2) H(1)
```

as returned by CGEQLF.

Syntax

Fortran specification:

```
use armpl_library

subroutine cungql(M, N, K, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cungql_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *k,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const armpl_singlecomplex_t *tau, armpl_singlecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix Q. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix Q. M >= N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. N >= K >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the (n-k+i)-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by CGEQLF in the last k columns of its array argument A. On exit, the M-by-N matrix Q.

LDA Input parameter.

LDA is INTEGER

The first dimension of the array A. LDA >= max(1, M).

TAU Input parameter.

TAU is COMPLEX

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by CGEQLF.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N). For optimum performance LWORK >= N*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument has an illegal value

Related Information

For this routine in other precisions, please see *zungql*. It also exists with a native C interface as *LAPACKE_cungql*.

4.3.45 cungqr

cungqr generates an M-by-N complex matrix Q with orthonormal columns, which is defined as the first N columns of a product of K elementary reflectors of order M

```
Q  =  H(1) H(2) . . . H(k)
```

as returned by CGEQRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine cungqr(M, N, K, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cungqr_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *k,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const armpl_singlecomplex_t *tau, armpl_singlecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix Q. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix Q. M >= N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. N >= K >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the i-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,. . . ,k, as returned by CGEQRF in the first k columns of its array argument A. On exit, the M-by-N matrix Q.

LDA Input parameter.

LDA is INTEGER

The first dimension of the array A. LDA >= max(1, M).

TAU Input parameter.

TAU is COMPLEX

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by CGEQRF.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N). For optimum performance LWORK >= N*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument has an illegal value

Related Information

For this routine in other precisions, please see *zungqr*. It also exists with a native C interface as *LAPACKE_cungqr*.

4.3.46 cungrq

cungrq generates an M-by-N complex matrix Q with orthonormal rows, which is defined as the last M rows of a product of K elementary reflectors of order N

```
Q  =  H(1)**H H(2)**H . . . H(k)**H
```

as returned by CGERQF.

Syntax

Fortran specification:

```
use armpl_library

subroutine cungrq(M, N, K, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cungrq_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *k,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const armpl_singlecomplex_t *tau, armpl_singlecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix Q. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix Q. N >= M.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. M >= K >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the (m-k+i)-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by CGERQF in the last k rows of its array argument A. On exit, the M-by-N matrix Q.

LDA Input parameter.

LDA is INTEGER

The first dimension of the array A. LDA >= max(1, M).

TAU Input parameter.

TAU is COMPLEX

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by CGERQF.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, M). For optimum performance LWORK >= M*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument has an illegal value

Related Information

For this routine in other precisions, please see *zungrq*. It also exists with a native C interface as *LAPACKE_cungrq*.

4.3.47 cunmlq

cunmlq overwrites the general complex M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'C': Q^H * C C * Q^H

where Q is a complex unitary matrix defined as the product of k elementary reflectors

```
Q = H(k)**H . . . H(2)**H H(1)**H
```

as returned by CGELQF. Q is of order M if SIDE = 'L' and of order N if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine cunmlq(SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void cunmlq_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const armpl_singlecomplex_t *tau, armpl_singlecomplex_t *c,
             const armpl_int_t *ldc, armpl_singlecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left; = 'R': apply Q or Q^H from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'C': Conjugate transpose, apply Q^H .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension. (LDA, M) if SIDE = 'L', (LDA, N) if SIDE = 'R' The i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by CGELQF in the first k rows of its array argument A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, K).

TAU Input parameter.

TAU is COMPLEX

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by CGELQF.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^H * C or C*Q^H or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, N); if SIDE = 'R', LWORK >= max(1, M). For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zunmlq*. It also exists with a native C interface as *LAPACKE_cunmlq*.

4.3.48 cunmql

cunmql overwrites the general complex M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'C': Q^H * C C * Q^H

where Q is a complex unitary matrix defined as the product of k elementary reflectors

```
Q = H(k) . . . H(2) H(1)
```

as returned by CGEQLF. Q is of order M if SIDE = 'L' and of order N if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine cunmql(SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void cunmql_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const armpl_singlecomplex_t *tau, armpl_singlecomplex_t *c,
             const armpl_int_t *ldc, armpl_singlecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left; = 'R': apply Q or Q^H from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'C': Transpose, apply Q^H .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, K). The i-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,. . . ,k, as returned by CGEQLF in the last k columns of its array argument A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDA >= max(1, M); if SIDE = 'R', LDA >= max(1, N).

TAU Input parameter.

TAU is COMPLEX

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by CGEQLF.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^H * C or C*Q^H or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, N); if SIDE = 'R', LWORK >= max(1, M). For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zunmql*. It also exists with a native C interface as *LAPACKE_cunmql*.

4.3.49 cunmqr

cunmqr overwrites the general complex M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'C': Q^H * C C * Q^H

where Q is a complex unitary matrix defined as the product of k elementary reflectors

```
Q = H(1) H(2) . . . H(k)
```

as returned by CGEQRF. Q is of order M if SIDE = 'L' and of order N if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine cunmqr(SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void cunmqr_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const armpl_singlecomplex_t *tau, armpl_singlecomplex_t *c,
             const armpl_int_t *ldc, armpl_singlecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left; = 'R': apply Q or Q^H from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'C': Conjugate transpose, apply Q^H .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, K). The i-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by CGEQRF in the first k columns of its array argument A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDA >= max(1, M); if SIDE = 'R', LDA >= max(1, N).

TAU Input parameter.

TAU is COMPLEX

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by CGEQRF.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^H * C or C*Q^H or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, N); if SIDE = 'R', LWORK >= max(1, M). For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zunmqr*. It also exists with a native C interface as *LAPACKE_cunmqr*.

4.3.50 cunmrq

cunmrq overwrites the general complex M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'C': Q^H * C C * Q^H

where Q is a complex unitary matrix defined as the product of k elementary reflectors

```
Q = H(1)**H H(2)**H . . . H(k)**H
```

as returned by CGERQF. Q is of order M if SIDE = 'L' and of order N if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine cunmrq(SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void cunmrq_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const armpl_singlecomplex_t *tau, armpl_singlecomplex_t *c,
             const armpl_int_t *ldc, armpl_singlecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left; = 'R': apply Q or Q^H from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'C': Transpose, apply Q^H .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension. (LDA, M) if SIDE = 'L', (LDA, N) if SIDE = 'R' The i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by CGERQF in the last k rows of its array argument A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, K).

TAU Input parameter.

TAU is COMPLEX

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by CGERQF.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^H * C or C*Q^H or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, N); if SIDE = 'R', LWORK >= max(1, M). For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zunmrq*. It also exists with a native C interface as *LAPACKE_cunmrq*.

4.3.51 cunmrz

cunmrz overwrites the general complex M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'C': Q^H * C C * Q^H

where Q is a complex unitary matrix defined as the product of k elementary reflectors

```
Q = H(1) H(2) . . . H(k)
```

as returned by CTZRZF. Q is of order M if SIDE = 'L' and of order N if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine cunmrz(SIDE, TRANS, M, N, K, L, A, LDA, TAU, C, LDC, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void cunmrz_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k, const armpl_int_t *l,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const armpl_singlecomplex_t *tau, armpl_singlecomplex_t *c,
             const armpl_int_t *ldc, armpl_singlecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left; = 'R': apply Q or Q^H from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'C': Conjugate transpose, apply Q^H .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

L Input parameter.

L is INTEGER

The number of columns of the matrix A containing the meaningful part of the Householder reflectors. If SIDE = 'L', M >= L >= 0, if SIDE = 'R', N >= L >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension. (LDA, M) if SIDE = 'L', (LDA, N) if SIDE = 'R' The i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by CTZRZF in the last k rows of its array argument A. A is modified by the routine but restored on exit.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, K).

TAU Input parameter.

TAU is COMPLEX

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by CTZRZF.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^H * C or C*Q^H or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, N); if SIDE = 'R', LWORK >= max(1, M). For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zunmrz*. It also exists with a native C interface as *LAPACKE_cunmrz*.

4.3.52 dgbtrf

dgbtrf computes an LU factorization of a real m-by-n band matrix A using partial pivoting with row interchanges.

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgbtrf(M, N, KL, KU, AB, LDAB, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void dgbtrf_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *kl, const armpl_int_t *ku, double *ab,
             const armpl_int_t *ldab, armpl_int_t *ipiv, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

AB Input and output parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows KL+1 to 2*KL+KU+1; rows 1 to KL of the array need not be set. The j-th column of A is stored in the j-th column of the array AB as follows: AB(kl+ku+1+i-j,j) = A(i,j) for max(1,j-ku)<=i<=min(m,j+kl)

On exit, details of the factorization: U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1. See below for further details.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= 2*KL+KU+1.

IPIV Output parameter.

IPIV is INTEGER array, dimension (min(M, N))

The pivot indices; for 1 <= i <= min(M, N), row i of the matrix was interchanged with row IPIV(i).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = +i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *cgbtrf*, *sgbtrf* and *zgbtrf*. It also exists with a native C interface as *LAPACKE_dgbtrf*.

4.3.53 dgelq

`dgelq` computes a LQ factorization of an M-by-N matrix A.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgelq(M, N, A, LDA, T, TSIZE, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgelq_(const armpl_int_t *m, const armpl_int_t *n, double *a,
            const armpl_int_t *lda, double *t, const armpl_int_t *tsize,
            double *work, const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

`M` is `INTEGER`

The number of rows of the matrix `A`. `M` >= 0.

N Input parameter.

`N` is `INTEGER`

The number of columns of the matrix `A`. `N` >= 0.

A Input and output parameter.

`A` is `DOUBLE PRECISION`

`A` is an array, dimension (`LDA`, `N`). On entry, the M-by-N matrix `A`. On exit, the elements on and below the diagonal of the array contain the M-by-min(`M`, `N`) lower trapezoidal matrix L (L is lower triangular if `M` <= N); the elements above the diagonal are used to store part of the data structure to represent Q.

LDA Input parameter.

`LDA` is `INTEGER`

The leading dimension of the array `A`. `LDA` >= max(1, `M`).

T Output parameter.

`T` is `DOUBLE PRECISION`

`T` is an array, dimension (MAX(5, `TSIZE`)). On exit, if `INFO` = 0, T(1) returns optimal (or either minimal or optimal, if query is assumed) `TSIZE`. See `TSIZE` for details. Remaining `T` contains part of the data structure used to represent Q. If one wants to apply or construct Q, then one needs to keep `T` (in addition to A) and pass it to further subroutines.

TSIZE Input parameter.

TSIZE is INTEGER

If TSIZE >= 5, the dimension of the array T. If TSIZE = -1 or -2, then a workspace query is assumed. The routine only calculates the sizes of the T and WORK arrays, returns these values as the first entries of the T and WORK arrays, and no error message related to T or WORK is issued by XERBLA. If TSIZE = -1, the routine calculates optimal size of T for the optimum performance and returns this value in T(1). If TSIZE = -2, the routine calculates minimal size of T and returns this value in T(1).

WORK Output parameter.

(workspace) DOUBLE PRECISION

(workspace) is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) contains optimal (or either minimal or optimal, if query was assumed) LWORK. See LWORK for details.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If LWORK = -1 or -2, then a workspace query is assumed. The routine only calculates the sizes of the T and WORK arrays, returns these values as the first entries of the T and WORK arrays, and no error message related to T or WORK is issued by XERBLA. If LWORK = -1, the routine calculates optimal size of WORK for the optimal performance and returns this value in WORK(1). If LWORK = -2, the routine calculates minimal size of WORK and returns this value in WORK(1).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgelq*, *sgelq* and *zgelq*. It also exists with a native C interface as *LAPACKE_dgelq*.

4.3.54 dgelqf

dgelqf computes an LQ factorization of a real M-by-N matrix A: A = L * Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgelqf(M, N, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgelqf_(const armpl_int_t *m, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, double *tau, double *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and below the diagonal of the array contain the m-by-min(m,n) lower trapezoidal matrix L (L is lower triangular if m <= n); the elements above the diagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, M). For optimum performance LWORK >= M*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgelqf*, *sgelqf* and *zgelqf*. It also exists with a native C interface as *LAPACKE_dgelqf*.

4.3.55 dgelqt

dgelqt computes a blocked LQ factorization of a real M-by-N matrix A using the compact WY representation of Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgelqt(M, N, MB, A, LDA, T, LDT, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgelqt_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *mb, double *a, const armpl_int_t *lda,
             double *t, const armpl_int_t *ldt, double *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

MB Input parameter.

MB is INTEGER

The block size to be used in the blocked QR. MIN(M, N) >= MB >= 1.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and below the diagonal of the array contain the M-by-MIN(M, N) lower trapezoidal matrix L (L is lower triangular if M <= N); the elements above the diagonal are the rows of V.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

T Output parameter.

T is DOUBLE PRECISION

T is an array, dimension (LDT,MIN(M, N)). The upper triangular block reflectors stored in compact form as a sequence of upper triangular blocks. See below for further details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= MB.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MB*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgelqt*, *sgelqt* and *zgelqt*.

4.3.56 dgemlq

dgemlq overwrites the general real M-by-N matrix C with SIDE = 'L' SIDE = 'R' TRANS = 'N': Q * C C * Q TRANS = 'T': Q^T * C C * Q^T where Q is a real orthogonal matrix defined as the product of blocked elementary reflectors computed by short wide LQ factorization (DGELQ)

Syntax

Fortran specification:

```
use armpl_library

subroutine dgemlq(SIDE, TRANS, M, N, K, A, LDA, T, TSIZE, C, LDC, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dgemlq_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k, const double *a,
             const armpl_int_t *lda, const double *t,
             const armpl_int_t *tsize, double *c, const armpl_int_t *ldc,
             double *work, const armpl_int_t *lwork, armpl_int_t *info,
             ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'T': Transpose, apply Q^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >=0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension. (LDA, M) if SIDE = 'L', (LDA, N) if SIDE = 'R' Part of the data structure to represent Q as returned by DGELQ.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, K).

T Input parameter.

T is DOUBLE PRECISION

T is an array, dimension (MAX(5, TSIZE)).. Part of the data structure to represent Q as returned by DGELQ.

TSIZE Input parameter.

TSIZE is INTEGER

The dimension of the array T. TSIZE >= 5.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

(workspace) DOUBLE PRECISION

(workspace) is an array, dimension (MAX(1, LWORK)) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If LWORK = -1, then a workspace query is assumed. The routine only calculates the size of the WORK array, returns this value as WORK(1), and no error message related to WORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgemlq*, *sgemlq* and *zgemlq*. It also exists with a native C interface as *LAPACKE_dgemlq*.

4.3.57 dgemlqt

dgemlqt overwrites the general real M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': Q C C Q TRANS = 'T': Q^T C C Q^T

where Q is a real orthogonal matrix defined as the product of K elementary reflectors:

```
Q = H(1) H(2) . . . H(K) = I - V T V**T
```

generated using the compact WY representation as returned by DGELQT.

Q is of order M if SIDE = 'L' and of order N if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgemlqt(SIDE, TRANS, M, N, K, MB, V, LDV, T, LDT, C, LDC, WORK,
                   INFO)
```

C specification:

```
#include "armpl.h"

void dgemlqt_(const char *side, const char *trans, const armpl_int_t *m,
              const armpl_int_t *n, const armpl_int_t *k,
              const armpl_int_t *mb, const double *v, const armpl_int_t *ldv,
              const double *t, const armpl_int_t *ldt, double *c,
              const armpl_int_t *ldc, double *work, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'C': Transpose, apply Q^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

MB Input parameter.

MB is INTEGER

The block size used for the storage of T. K >= MB >= 1. This must be the same value of MB used to generate T in DGELQT.

V Input parameter.

V is DOUBLE PRECISION

V is an array, dimension. (LDV, M) if SIDE = 'L', (LDV, N) if SIDE = 'R' The i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by DGELQT in the first K rows of its array argument A.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. LDV >= max(1, K).

T Input parameter.

T is DOUBLE PRECISION

T is an array, dimension (LDT, K). The upper triangular factors of the block reflectors as returned by DGELQT, stored as a MB-by-K matrix.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= MB.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q C, Q^T C, C Q^T or C Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is DOUBLE PRECISION array. The dimension of

WORK is N* MB if SIDE = 'L', or M* MB if SIDE = 'R'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgemlqt*, *sgemlqt* and *zgemlqt*.

4.3.58 dgemqr

dgemqr overwrites the general real M-by-N matrix C with SIDE = 'L' SIDE = 'R' TRANS = 'N': Q * C C * Q TRANS = 'T': Q^T * C C * Q^T where Q is a real orthogonal matrix defined as the product of blocked elementary reflectors computed by tall skinny QR factorization (DGEQR)

Syntax

Fortran specification:

```
use armpl_library

subroutine dgemqr(SIDE, TRANS, M, N, K, A, LDA, T, TSIZE, C, LDC, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dgemqr_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k, const double *a,
             const armpl_int_t *lda, const double *t,
             const armpl_int_t *tsize, double *c, const armpl_int_t *ldc,
             double *work, const armpl_int_t *lwork, armpl_int_t *info,
             ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'T': Transpose, apply Q^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >=0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, K). Part of the data structure to represent Q as returned by DGEQR.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDA >= max(1, M); if SIDE = 'R', LDA >= max(1, N).

T Input parameter.

T is DOUBLE PRECISION

T is an array, dimension (MAX(5, TSIZE)).. Part of the data structure to represent Q as returned by DGEQR.

TSIZE Input parameter.

TSIZE is INTEGER

The dimension of the array T. TSIZE >= 5.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

(workspace) DOUBLE PRECISION

(workspace) is an array, dimension (MAX(1, LWORK)) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If LWORK = -1, then a workspace query is assumed. The routine only calculates the size of the WORK array, returns this value as WORK(1), and no error message related to WORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgemqr*, *sgemqr* and *zgemqr*. It also exists with a native C interface as *LAPACKE_dgemqr*.

4.3.59 dgemqrt

dgemqrt overwrites the general real M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': Q C C Q TRANS = 'T': Q^T C C Q^T

where Q is a real orthogonal matrix defined as the product of K elementary reflectors:

```
Q = H(1) H(2) . . . H(K) = I - V T V**T
```

generated using the compact WY representation as returned by DGEQRT.

Q is of order M if SIDE = 'L' and of order N if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgemqrt(SIDE, TRANS, M, N, K, NB, V, LDV, T, LDT, C, LDC, WORK,
                   INFO)
```

C specification:

```
#include "armpl.h"

void dgemqrt_(const char *side, const char *trans, const armpl_int_t *m,
              const armpl_int_t *n, const armpl_int_t *k,
              const armpl_int_t *nb, const double *v, const armpl_int_t *ldv,
              const double *t, const armpl_int_t *ldt, double *c,
              const armpl_int_t *ldc, double *work, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'C': Transpose, apply Q^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

NB Input parameter.

NB is INTEGER

The block size used for the storage of T. K >= NB >= 1. This must be the same value of NB used to generate T in CGEQRT.

V Input parameter.

V is DOUBLE PRECISION

V is an array, dimension (LDV, K). The i-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,. . . ,k, as returned by CGEQRT in the first K columns of its array argument A.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. If SIDE = 'L', LDA >= max(1, M); if SIDE = 'R', LDA >= max(1, N).

T Input parameter.

T is DOUBLE PRECISION

T is an array, dimension (LDT, K). The upper triangular factors of the block reflectors as returned by CGEQRT, stored as a NB-by-N matrix.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= NB.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q C, Q^T C, C Q^T or C Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is DOUBLE PRECISION array. The dimension of

WORK is N* NB if SIDE = 'L', or M* NB if SIDE = 'R'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgemqrt*, *sgemqrt* and *zgemqrt*. It also exists with a native C interface as *LAPACKE_dgemqrt*.

4.3.60 dgeqlf

dgeqlf computes a QL factorization of a real M-by-N matrix A: A = Q * L.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgeqlf(M, N, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgeqlf_(const armpl_int_t *m, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, double *tau, double *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, if m >= n, the lower triangle of the subarray A(m-n+1:m,1:n) contains the N-by-N lower triangular matrix L; if m <= n, the elements on and below the (n-m)-th superdiagonal contain the M-by-N lower trapezoidal matrix L; the remaining elements, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N). For optimum performance LWORK >= N*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgeqlf*, *sgeqlf* and *zgeqlf*. It also exists with a native C interface as *LAPACKE_dgeqlf*.

4.3.61 dgeqp3

`dgeqp3` computes a QR factorization with column pivoting of a matrix A: A*P = Q*R using Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgeqp3(M, N, A, LDA, JPVT, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgeqp3_(const armpl_int_t *m, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, armpl_int_t *jpvt, double *tau,
             double *work, const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the upper triangle of the array contains the min(M, N)-by-N upper trapezoidal matrix R; the elements below the diagonal, together with the array TAU, represent the orthogonal matrix Q as a product of min(M, N) elementary reflectors.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

JPVT Input and output parameter.

JPVT is INTEGER array, dimension (N)

On entry, if JPVT(J).ne.0, the J-th column of A is permuted to the front of A*P (a leading column); if JPVT(J)=0, the J-th column of A is a free column. On exit, if JPVT(J)=K, then the J-th column of A*P was the the K-th column of A.

TAU Output parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO=0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= 3*N+1. For optimal performance LWORK >= 2*N+(N+1)*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cgeqp3*, *sgeqp3* and *zgeqp3*. It also exists with a native C interface as *LAPACKE_dgeqp3*.

4.3.62 dgeqr

dgeqr computes a QR factorization of an M-by-N matrix A.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgeqr(M, N, A, LDA, T, TSIZE, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgeqr_(const armpl_int_t *m, const armpl_int_t *n, double *a,
            const armpl_int_t *lda, double *t, const armpl_int_t *tsize,
            double *work, const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the min(M, N)-by-N upper trapezoidal matrix R (R is upper triangular if M >= N); the elements below the diagonal are used to store part of the data structure to represent Q.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

T Output parameter.

T is DOUBLE PRECISION

T is an array, dimension (MAX(5, TSIZE)). On exit, if INFO = 0, T(1) returns optimal (or either minimal or optimal, if query is assumed) TSIZE. See TSIZE for details. Remaining T contains part of the data structure used to represent Q. If one wants to apply or construct Q, then one needs to keep T (in addition to A) and pass it to further subroutines.

TSIZE Input parameter.

TSIZE is INTEGER

If TSIZE >= 5, the dimension of the array T. If TSIZE = -1 or -2, then a workspace query is assumed. The routine only calculates the sizes of the T and WORK arrays, returns these values as the first entries of the T and WORK arrays, and no error message related to T or WORK is issued by XERBLA. If TSIZE = -1, the routine calculates optimal size of T for the optimum performance and returns this value in T(1). If TSIZE = -2, the routine calculates minimal size of T and returns this value in T(1).

WORK Output parameter.

(workspace) DOUBLE PRECISION

(workspace) is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) contains optimal (or either minimal or optimal, if query was assumed) LWORK. See LWORK for details.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If LWORK = -1 or -2, then a workspace query is assumed. The routine only calculates the sizes of the T and WORK arrays, returns these values as the first entries of the T and WORK arrays, and no error message related to T or WORK is issued by XERBLA. If LWORK = -1, the routine calculates optimal size of WORK for the optimal performance and returns this value in WORK(1). If LWORK = -2, the routine calculates minimal size of WORK and returns this value in WORK(1).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgeqr*, *sgeqr* and *zgeqr*. It also exists with a native C interface as *LAPACKE_dgeqr*.

4.3.63 dgeqrf

`dgeqrf` computes a QR factorization of a real M-by-N matrix A: A = Q * R.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgeqrf(M, N, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgeqrf_(const armpl_int_t *m, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, double *tau, double *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the min(M, N)-by-N upper trapezoidal matrix R (R is upper triangular if m >= n); the elements below the diagonal, with the array TAU, represent the orthogonal matrix Q as a product of min(m,n) elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N). For optimum performance LWORK >= N*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgeqrf*, *sgeqrf* and *zgeqrf*. It also exists with a native C interface as *LAPACKE_dgeqrf*.

4.3.64 dgeqrfp

dgeqrfp computes a QR factorization of a real M-by-N matrix A: A = Q * R. The diagonal entries of R are nonnegative.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgeqrfp(M, N, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgeqrfp_(const armpl_int_t *m, const armpl_int_t *n, double *a,
              const armpl_int_t *lda, double *tau, double *work,
              const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the min(M, N)-by-N upper trapezoidal matrix R (R is upper triangular if m >= n). The diagonal entries of R are nonnegative; the elements below the diagonal, with the array TAU, represent the orthogonal matrix Q as a product of min(m,n) elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N). For optimum performance LWORK >= N*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgeqrfp*, *sgeqrfp* and *zgeqrfp*. It also exists with a native C interface as *LAPACKE_dgeqrfp*.

4.3.65 dgeqrt

dgeqrt computes a blocked QR factorization of a real M-by-N matrix A using the compact WY representation of Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgeqrt(M, N, NB, A, LDA, T, LDT, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgeqrt_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *nb, double *a, const armpl_int_t *lda,
             double *t, const armpl_int_t *ldt, double *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

NB Input parameter.

NB is INTEGER

The block size to be used in the blocked QR. MIN(M, N) >= NB >= 1.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the min(M, N)-by-N upper trapezoidal matrix R (R is upper triangular if M >= N); the elements below the diagonal are the columns of V.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

T Output parameter.

T is DOUBLE PRECISION

T is an array, dimension (LDT,MIN(M, N)). The upper triangular block reflectors stored in compact form as a sequence of upper triangular blocks. See below for further details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= NB.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (NB*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgeqrt*, *sgeqrt* and *zgeqrt*. It also exists with a native C interface as *LAPACKE_dgeqrt*.

4.3.66 dgerqf

dgerqf computes an RQ factorization of a real M-by-N matrix A: A = R * Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgerqf(M, N, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgerqf_(const armpl_int_t *m, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, double *tau, double *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, if m <= n, the upper triangle of the subarray A(1:m,n-m+1:n) contains the M-by-M upper triangular matrix R; if m >= n, the elements on and above the (m-n)-th subdiagonal contain the M-by-N upper trapezoidal matrix R; the remaining elements, with the array TAU, represent the orthogonal matrix Q as a product of min(m,n) elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, M). For optimum performance LWORK >= M*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgerqf*, *sgerqf* and *zgerqf*. It also exists with a native C interface as *LAPACKE_dgerqf*.

4.3.67 dgetrf

dgetrf computes an LU factorization of a general M-by-N matrix A using partial pivoting with row interchanges.

The factorization has the form

```
A = P * L * U
```

where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if m > n), and U is upper triangular (upper trapezoidal if m < n).

This is the right-looking Level 3 BLAS version of the algorithm.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgetrf(M, N, A, LDA, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void dgetrf_(const armpl_int_t *m, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, armpl_int_t *ipiv, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the M-by-N matrix to be factored. On exit, the factors L and U from the factorization A = P*L*U; the unit diagonal elements of L are not stored.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

IPIV Output parameter.

IPIV is INTEGER array, dimension (min(M, N))

The pivot indices; for 1 <= i <= min(M, N), row i of the matrix was interchanged with row IPIV(i).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *cgetrf*, *sgetrf* and *zgetrf*. It also exists with a native C interface as *LAPACKE_dgetrf*.

4.3.68 dgetrf2

dgetrf2 computes an LU factorization of a general M-by-N matrix A using partial pivoting with row interchanges.

The factorization has the form

```
A = P * L * U
```

where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if m > n), and U is upper triangular (upper trapezoidal if m < n).

This is the recursive version of the algorithm. It divides the matrix into four submatrices:

```
    [  A11 | A12  ]  where A11 is n1 by n1 and A22 is n2 by n2
A = [ -----|-----  ]  with n1 = min(m,n)/2
    [  A21 | A22  ]       n2 = n-n1

                                  [ A11 ]
```

The subroutine calls itself to factor [—],

```
                      [ A12 ]
[ A12 ]
```

do the swaps on [—], solve A12, update A22,

```
[ A22 ]
```

then calls itself to factor A22 and do the swaps on A21.

Syntax

Fortran specification:

```
use armpl_library

recursive subroutine dgetrf2(M, N, A, LDA, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void dgetrf2_(const armpl_int_t *m, const armpl_int_t *n, double *a,
              const armpl_int_t *lda, armpl_int_t *ipiv, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the M-by-N matrix to be factored. On exit, the factors L and U from the factorization A = P*L*U; the unit diagonal elements of L are not stored.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

IPIV Output parameter.

IPIV is INTEGER array, dimension (min(M, N))

The pivot indices; for 1 <= i <= min(M, N), row i of the matrix was interchanged with row IPIV(i).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *cgetrf2*, *sgetrf2* and *zgetrf2*. It also exists with a native C interface as *LAPACKE_dgetrf2*.

4.3.69 dggqrf

dggqrf computes a generalized QR factorization of an N-by-M matrix A and an N-by-P matrix B:

```
A = Q*R,        B = Q*T*Z,
```

where Q is an N-by-N orthogonal matrix, Z is a P-by-P orthogonal matrix, and R and T assume one of the forms:

if N >= M, R = (R11) M , or if N < M, R = (R11 R12) N,

```
(  0  ) N-M                          N   M-N
   M
```

where R11 is upper triangular, and

if N <= P, T = (0 T12) N, or if N > P, T = (T11) N-P,

```
P-N  N                                ( T21 ) P
                                        P
```

where T12 or T21 is upper triangular.

In particular, if B is square and nonsingular, the GQR factorization of A and B implicitly gives the QR factorization of inv(B)*A:

```
inv(B)*A = Z**T*(inv(T)*R)
```

where inv(B) denotes the inverse of the matrix B, and Z^T denotes the transpose of the matrix Z.

Syntax

Fortran specification:

```
use armpl_library

subroutine dggqrf(N, M, P, A, LDA, TAUA, B, LDB, TAUB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dggqrf_(const armpl_int_t *n, const armpl_int_t *m, const armpl_int_t *p,
             double *a, const armpl_int_t *lda, double *taua, double *b,
             const armpl_int_t *ldb, double *taub, double *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The number of rows of the matrices A and B. N >= 0.

M Input parameter.

M is INTEGER

The number of columns of the matrix A. M >= 0.

P Input parameter.

P is INTEGER

The number of columns of the matrix B. P >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, M). On entry, the N-by-M matrix A. On exit, the elements on and above the diagonal of the array contain the min(N, M)-by-M upper trapezoidal matrix R (R is upper triangular if N >= M); the elements below the diagonal, with the array TAUA, represent the orthogonal matrix Q as a product of min(N, M) elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TAUA Output parameter.

TAUA is DOUBLE PRECISION

TAUA is an array, dimension (min(N, M)). The scalar factors of the elementary reflectors which represent the orthogonal matrix Q (see Further Details).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, P). On entry, the N-by-P matrix B. On exit, if N <= P, the upper triangle of the subarray B(1:N,P-N+1:P) contains the N-by-N upper triangular matrix T; if N > P, the elements on and above the (N-P)-th subdiagonal contain the N-by-P upper trapezoidal matrix T; the remaining elements, with the array TAUB, represent the orthogonal matrix Z as a product of elementary reflectors (see Further Details).

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

TAUB Output parameter.

TAUB is DOUBLE PRECISION

TAUB is an array, dimension (min(N, P)). The scalar factors of the elementary reflectors which represent the orthogonal matrix Z (see Further Details).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N, M, P). For optimum performance LWORK >= max(N, M, P)*max(NB1,NB2,NB3), where NB1 is the optimal blocksize for the QR factorization of an N-by-M matrix, NB2 is the optimal blocksize for the RQ factorization of an N-by-P matrix, and NB3 is the optimal blocksize for a call of DORMQR.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cggqrf*, *sggqrf* and *zggqrf*. It also exists with a native C interface as *LAPACKE_dggqrf*.

4.3.70 dggrqf

dggrqf computes a generalized RQ factorization of an M-by-N matrix A and a P-by-N matrix B:

```
A = R*Q,        B = Z*T*Q,
```

where Q is an N-by-N orthogonal matrix, Z is a P-by-P orthogonal matrix, and R and T assume one of the forms:

if M <= N, R = (0 R12) M, or if M > N, R = (R11) M-N,

```
N-M  M                              ( R21 ) N
                                      N
```

where R12 or R21 is upper triangular, and

if P >= N, T = (T11) N , or if P < N, T = (T11 T12) P,

```
(  0  ) P-N                           P   N-P
  N
```

where T11 is upper triangular.

In particular, if B is square and nonsingular, the GRQ factorization of A and B implicitly gives the RQ factorization of A*inv(B):

```
A*inv(B) = (R*inv(T))*Z**T
```

where inv(B) denotes the inverse of the matrix B, and Z^T denotes the transpose of the matrix Z.

Syntax

Fortran specification:

```
use armpl_library

subroutine dggrqf(M, P, N, A, LDA, TAUA, B, LDB, TAUB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dggrqf_(const armpl_int_t *m, const armpl_int_t *p, const armpl_int_t *n,
             double *a, const armpl_int_t *lda, double *taua, double *b,
             const armpl_int_t *ldb, double *taub, double *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

P Input parameter.

P is INTEGER

The number of rows of the matrix B. P >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrices A and B. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, if M <= N, the upper triangle of the subarray A(1:M,N-M+1:N) contains the M-by-M upper triangular matrix R; if M > N, the elements on and above the (M-N)-th subdiagonal contain the M-by-N upper trapezoidal matrix R; the remaining elements, with the array TAUA, represent the orthogonal matrix Q as a product of elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAUA Output parameter.

TAUA is DOUBLE PRECISION

TAUA is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors which represent the orthogonal matrix Q (see Further Details).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). On entry, the P-by-N matrix B. On exit, the elements on and above the diagonal of the array contain the min(P, N)-by-N upper trapezoidal matrix T (T is upper triangular if P >= N); the elements below the diagonal, with the array TAUB, represent the orthogonal matrix Z as a product of elementary reflectors (see Further Details).

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, P).

TAUB Output parameter.

TAUB is DOUBLE PRECISION

TAUB is an array, dimension (min(P, N)). The scalar factors of the elementary reflectors which represent the orthogonal matrix Z (see Further Details).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N, M, P). For optimum performance LWORK >= max(N, M, P)*max(NB1,NB2,NB3), where NB1 is the optimal blocksize for the RQ factorization of an M-by-N matrix, NB2 is the optimal blocksize for the QR factorization of a P-by-N matrix, and NB3 is the optimal blocksize for a call of DORMRQ.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INF0= -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cggrqf*, *sggrqf* and *zggrqf*. It also exists with a native C interface as *LAPACKE_dggrqf*.

4.3.71 dgttrf

dgttrf computes an LU factorization of a real tridiagonal matrix A using elimination with partial pivoting and row interchanges.

The factorization has the form

```
A = L * U
```

where L is a product of permutation and unit lower bidiagonal matrices and U is upper triangular with nonzeros in only the main diagonal and first two superdiagonals.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgttrf(N, DL, D, DU, DU2, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void dgttrf_(const armpl_int_t *n, double *dl, double *d, double *du,
             double *du2, armpl_int_t *ipiv, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A.

DL Input and output parameter.

DL is DOUBLE PRECISION

DL is an array, dimension (N-1). On entry, DL must contain the (n-1) sub-diagonal elements of A.

On exit, DL is overwritten by the (n-1) multipliers that define the matrix L from the LU factorization of A.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry, D must contain the diagonal elements of A.

On exit, D is overwritten by the n diagonal elements of the upper triangular matrix U from the LU factorization of A.

DU Input and output parameter.

DU is DOUBLE PRECISION

DU is an array, dimension (N-1). On entry, DU must contain the (n-1) super-diagonal elements of A.

On exit, DU is overwritten by the (n-1) elements of the first super-diagonal of U.

DU2 Output parameter.

DU2 is DOUBLE PRECISION

DU2 is an array, dimension (N-2). On exit, DU2 is overwritten by the (n-2) elements of the second super-diagonal of U.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices; for 1 <= i <= n, row i of the matrix was interchanged with row IPIV(i). IPIV(i) will always be either i or i+1; IPIV(i) = i indicates a row interchange was not required.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value > 0: if INFO = k, U(k,k) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *cgttrf*, *sgttrf* and *zgttrf*. It also exists with a native C interface as *LAPACKE_dgttrf*.

4.3.72 dorglq

dorglq generates an M-by-N real matrix Q with orthonormal rows, which is defined as the first M rows of a product of K elementary reflectors of order N

```
Q  =  H(k) . . . H(2) H(1)
```

as returned by DGELQF.

Syntax

Fortran specification:

```
use armpl_library

subroutine dorglq(M, N, K, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dorglq_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *k,
             double *a, const armpl_int_t *lda, const double *tau,
             double *work, const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix Q. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix Q. N >= M.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. M >= K >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,. . . ,k, as returned by DGELQF in the first k rows of its array argument A. On exit, the M-by-N matrix Q.

LDA Input parameter.

LDA is INTEGER

The first dimension of the array A. LDA >= max(1, M).

TAU Input parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by DGELQF.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, M). For optimum performance LWORK >= M*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument has an illegal value

Related Information

For this routine in other precisions, please see *sorglq*. It also exists with a native C interface as *LAPACKE_dorglq*.

4.3.73 dorgql

dorgql generates an M-by-N real matrix Q with orthonormal columns, which is defined as the last N columns of a product of K elementary reflectors of order M

```
Q  =  H(k) . . . H(2) H(1)
```

as returned by DGEQLF.

Syntax

Fortran specification:

```
use armpl_library

subroutine dorgql(M, N, K, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dorgql_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *k,
             double *a, const armpl_int_t *lda, const double *tau,
             double *work, const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix Q. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix Q. M >= N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. N >= K >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the (n-k+i)-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by DGEQLF in the last k columns of its array argument A. On exit, the M-by-N matrix Q.

LDA Input parameter.

LDA is INTEGER

The first dimension of the array A. LDA >= max(1, M).

TAU Input parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by DGEQLF.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N). For optimum performance LWORK >= N*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument has an illegal value

Related Information

For this routine in other precisions, please see *sorgql*. It also exists with a native C interface as *LAPACKE_dorgql*.

4.3.74 dorgqr

dorgqr generates an M-by-N real matrix Q with orthonormal columns, which is defined as the first N columns of a product of K elementary reflectors of order M

```
Q  =  H(1) H(2) . . . H(k)
```

as returned by DGEQRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine dorgqr(M, N, K, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dorgqr_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *k,
             double *a, const armpl_int_t *lda, const double *tau,
             double *work, const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix Q. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix Q. M >= N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. N >= K >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the i-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by DGEQRF in the first k columns of its array argument A. On exit, the M-by-N matrix Q.

LDA Input parameter.

LDA is INTEGER

The first dimension of the array A. LDA >= max(1, M).

TAU Input parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by DGEQRF.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N). For optimum performance LWORK >= N*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument has an illegal value

Related Information

For this routine in other precisions, please see *sorgqr*. It also exists with a native C interface as *LAPACKE_dorgqr*.

4.3.75 dorgrq

dorgrq generates an M-by-N real matrix Q with orthonormal rows, which is defined as the last M rows of a product of K elementary reflectors of order N

```
Q  =  H(1) H(2) . . . H(k)
```

as returned by DGERQF.

Syntax

Fortran specification:

```
use armpl_library

subroutine dorgrq(M, N, K, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dorgrq_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *k,
             double *a, const armpl_int_t *lda, const double *tau,
             double *work, const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix Q. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix Q. N >= M.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. M >= K >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the (m-k+i)-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by DGERQF in the last k rows of its array argument A. On exit, the M-by-N matrix Q.

LDA Input parameter.

LDA is INTEGER

The first dimension of the array A. LDA >= max(1, M).

TAU Input parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by DGERQF.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, M). For optimum performance LWORK >= M*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument has an illegal value

Related Information

For this routine in other precisions, please see *sorgrq*. It also exists with a native C interface as *LAPACKE_dorgrq*.

4.3.76 dormlq

dormlq overwrites the general real M-by-N matrix C with

```
SIDE = 'L'      SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'T': Q^T * C C * Q^T

where Q is a real orthogonal matrix defined as the product of k elementary reflectors

```
Q = H(k) . . . H(2) H(1)
```

as returned by DGELQF. Q is of order M if SIDE = 'L' and of order N if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine dormlq(SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dormlq_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k, const double *a,
             const armpl_int_t *lda, const double *tau, double *c,
             const armpl_int_t *ldc, double *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'T': Transpose, apply Q^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension. (LDA, M) if SIDE = 'L', (LDA, N) if SIDE = 'R' The i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by DGELQF in the first k rows of its array argument A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, K).

TAU Input parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by DGELQF.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, N); if SIDE = 'R', LWORK >= max(1, M). For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *sormlq*. It also exists with a native C interface as *LAPACKE_dormlq*.

4.3.77 dormql

`dormql` overwrites the general real M-by-N matrix C with

```
SIDE = 'L'      SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'T': Q^T * C C * Q^T

where Q is a real orthogonal matrix defined as the product of k elementary reflectors

```
Q = H(k) . . . H(2) H(1)
```

as returned by DGEQLF. Q is of order M if SIDE = 'L' and of order N if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine dormql(SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dormql_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k, const double *a,
             const armpl_int_t *lda, const double *tau, double *c,
             const armpl_int_t *ldc, double *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'T': Transpose, apply Q^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, K). The i-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by DGEQLF in the last k columns of its array argument A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDA >= max(1, M); if SIDE = 'R', LDA >= max(1, N).

TAU Input parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by DGEQLF.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, N); if SIDE = 'R', LWORK >= max(1, M). For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *sormql*. It also exists with a native C interface as *LAPACKE_dormql*.

4.3.78 dormqr

dormqr overwrites the general real M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'T': Q^T * C C * Q^T

where Q is a real orthogonal matrix defined as the product of k elementary reflectors

```
Q = H(1) H(2) . . . H(k)
```

as returned by DGEQRF. Q is of order M if SIDE = 'L' and of order N if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine dormqr(SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dormqr_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k, const double *a,
             const armpl_int_t *lda, const double *tau, double *c,
             const armpl_int_t *ldc, double *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'T': Transpose, apply Q^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, K). The i-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,. . . ,k, as returned by DGEQRF in the first k columns of its array argument A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDA >= max(1, M); if SIDE = 'R', LDA >= max(1, N).

TAU Input parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by DGEQRF.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, N); if SIDE = 'R', LWORK >= max(1, M). For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *sormqr*. It also exists with a native C interface as *LAPACKE_dormqr*.

4.3.79 dormrq

dormrq overwrites the general real M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'T': Q^T * C C * Q^T

where Q is a real orthogonal matrix defined as the product of k elementary reflectors

```
Q = H(1) H(2) . . . H(k)
```

as returned by DGERQF. Q is of order M if SIDE = 'L' and of order N if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine dormrq(SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dormrq_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k, const double *a,
             const armpl_int_t *lda, const double *tau, double *c,
             const armpl_int_t *ldc, double *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'T': Transpose, apply Q^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension. (LDA, M) if SIDE = 'L', (LDA, N) if SIDE = 'R' The i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by DGERQF in the last k rows of its array argument A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, K).

TAU Input parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by DGERQF.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, N); if SIDE = 'R', LWORK >= max(1, M). For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *sormrq*. It also exists with a native C interface as *LAPACKE_dormrq*.

4.3.80 dormrz

dormrz overwrites the general real M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'T': Q^T * C C * Q^T

where Q is a real orthogonal matrix defined as the product of k elementary reflectors

```
Q = H(1) H(2) . . . H(k)
```

as returned by DTZRZF. Q is of order M if SIDE = 'L' and of order N if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine dormrz(SIDE, TRANS, M, N, K, L, A, LDA, TAU, C, LDC, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dormrz_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k, const armpl_int_t *l,
             const double *a, const armpl_int_t *lda, const double *tau,
             double *c, const armpl_int_t *ldc, double *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'T': Transpose, apply Q^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

L Input parameter.

L is INTEGER

The number of columns of the matrix A containing the meaningful part of the Householder reflectors. If SIDE = 'L', M >= L >= 0, if SIDE = 'R', N >= L >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension. (LDA, M) if SIDE = 'L', (LDA, N) if SIDE = 'R' The i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by DTZRZF in the last k rows of its array argument A. A is modified by the routine but restored on exit.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, K).

TAU Input parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by DTZRZF.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^H * C or C*Q^H or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, N); if SIDE = 'R', LWORK >= max(1, M). For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *sormrz*. It also exists with a native C interface as *LAPACKE_dormrz*.

4.3.81 dpbtrf

dpbtrf computes the Cholesky factorization of a real symmetric positive definite band matrix A.

The factorization has the form

```
A = U**T * U,  if UPLO = 'U', or
A = L  * L**T,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is lower triangular.

Syntax

Fortran specification:

```
use armpl_library

subroutine dpbtrf(UPLO, N, KD, AB, LDAB, INFO)
```

C specification:

```
#include "armpl.h"

void dpbtrf_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
             double *ab, const armpl_int_t *ldab, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, if INFO = 0, the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T of the band matrix A, in the same storage format as A.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i is not positive definite, and the factorization could not be completed.

Related Information

For this routine in other precisions, please see *cpbtrf*, *spbtrf* and *zpbtrf*. It also exists with a native C interface as *LAPACKE_dpbtrf*.

4.3.82 dpftrf

dpftrf computes the Cholesky factorization of a real symmetric positive definite matrix A.

The factorization has the form

```
A = U**T * U,  if UPLO = 'U', or
A = L  * L**T,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is lower triangular.

This is the block version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine dpftrf(TRANSR, UPLO, N, A, INFO)
```

C specification:

```
#include "armpl.h"

void dpftrf_(const char *transr, const char *uplo, const armpl_int_t *n,
             double *a, armpl_int_t *info, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': The Normal TRANSR of RFP A is stored; = 'T': The Transpose TRANSR of RFP A is stored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of RFP A is stored; = 'L': Lower triangle of RFP A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (N*(N+1)/2);. On entry, the symmetric matrix A in RFP format. RFP format is described by TRANSR, UPLO, and N as follows: If TRANSR = 'N' then RFP A is (0:N,0:k-1) when N is even; k=N/2. RFP A is (0:N-1,0:k) when N is odd; k=N/2. IF TRANSR = 'T' then RFP is the transpose of RFP A as defined when TRANSR = 'N'. The contents of RFP A are defined by UPLO as follows: If UPLO = 'U' the RFP A contains the NT elements of upper packed A. If UPLO = 'L' the RFP A contains the elements of lower packed A. The LDA of RFP A is (N+1)/2 when TRANSR = 'T'. When TRANSR is 'N' the LDA is N+1 when N is even and N is odd. See the Note below for more details.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization RFP A = U^T *U or RFP A = L*L^T .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i is not positive definite, and the factorization could not be completed.

Related Information

For this routine in other precisions, please see *cpftrf*, *spftrf* and *zpftrf*. It also exists with a native C interface as *LAPACKE_dpftrf*.

4.3.83 dpotrf

dpotrf computes the Cholesky factorization of a real symmetric positive definite matrix A.

The factorization has the form

```
A = U**T * U,  if UPLO = 'U', or
A = L  * L**T,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is lower triangular.

This is the block version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine dpotrf(UPLO, N, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void dpotrf_(const char *uplo, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T .

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i is not positive definite, and the factorization could not be completed.

Related Information

For this routine in other precisions, please see *cpotrf*, *spotrf* and *zpotrf*. It also exists with a native C interface as *LAPACKE_dpotrf*.

4.3.84 dpotrf2

dpotrf2 computes the Cholesky factorization of a real symmetric positive definite matrix A using the recursive algorithm.

The factorization has the form

```
A = U**T * U,  if UPLO = 'U', or
A = L  * L**T,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is lower triangular.

This is the recursive version of the algorithm. It divides the matrix into four submatrices:

```
    [  A11 | A12  ]  where A11 is n1 by n1 and A22 is n2 by n2
A = [ -----|-----  ]  with n1 = n/2
    [  A21 | A22  ]       n2 = n-n1
```

The subroutine calls itself to factor A11. Update and scale A21 or A12, update A22 then calls itself to factor A22.

Syntax

Fortran specification:

```
use armpl_library

recursive subroutine dpotrf2(UPLO, N, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void dpotrf2_(const char *uplo, const armpl_int_t *n, double *a,
              const armpl_int_t *lda, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T .

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i is not positive definite, and the factorization could not be completed.

Related Information

For this routine in other precisions, please see *cpotrf2*, *spotrf2* and *zpotrf2*. It also exists with a native C interface as *LAPACKE_dpotrf2*.

4.3.85 dpptrf

dpptrf computes the Cholesky factorization of a real symmetric positive definite matrix A stored in packed format.

The factorization has the form

```
A = U**T * U,  if UPLO = 'U', or
A = L  * L**T,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is lower triangular.

Syntax

Fortran specification:

```
use armpl_library

subroutine dpptrf(UPLO, N, AP, INFO)
```

C specification:

```
#include "armpl.h"

void dpptrf_(const char *uplo, const armpl_int_t *n, double *ap,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is DOUBLE PRECISION

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n. See below for further details.

On exit, if INFO = 0, the triangular factor U or L from the Cholesky factorization $A = U^T * U$ or $A = L * L^T$, in the same storage format as A.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i is not positive definite, and the factorization could not be completed.

Related Information

For this routine in other precisions, please see *cpptrf*, *spptrf* and *zpptrf*. It also exists with a native C interface as *LAPACKE_dpptrf*.

4.3.86 dpstrf

dpstrf computes the Cholesky factorization with complete pivoting of a real symmetric positive semidefinite matrix A.

The factorization has the form

```
P**T * A * P = U**T * U ,  if UPLO = 'U',
P**T * A * P = L  * L**T,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is lower triangular, and P is stored as vector PIV.

This algorithm does not attempt to check that A is positive semidefinite. This version of the algorithm calls level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine dpstrf(UPLO, N, A, LDA, PIV, RANK, TOL, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dpstrf_(const char *uplo, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, armpl_int_t *piv, armpl_int_t *rank,
             const double *tol, double *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored. = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading n by n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n by n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization as above.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

PIV Output parameter.

PIV is INTEGER array, dimension (N)

PIV is such that the nonzero entries are P(PIV(K), K) = 1.

RANK Output parameter.

RANK is INTEGER

The rank of A given by the number of steps the algorithm completed.

TOL Input parameter.

TOL is DOUBLE PRECISION

User defined tolerance. If TOL < 0, then N*U*MAX(A(K,K)) will be used. The algorithm terminates at the (K-1)st step if the pivot <= TOL.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (2*N). Work space.

INFO Output parameter.

INFO is INTEGER

< 0: If INFO = -K, the K-th argument had an illegal value, = 0: algorithm completed successfully, and > 0: the matrix A is either rank deficient with computed rank as returned in RANK, or is not positive semidefinite. See Section 7 of LAPACK Working Note #161 for further information.

Related Information

For this routine in other precisions, please see *cpstrf*, *spstrf* and *zpstrf*. It also exists with a native C interface as *LAPACKE_dpstrf*.

4.3.87 dpttrf

`dpttrf` computes the L*D*L^T factorization of a real symmetric positive definite tridiagonal matrix A. The factorization may also be regarded as having the form A = U^T *D*U.

Syntax

Fortran specification:

```
use armpl_library

subroutine dpttrf(N, D, E, INFO)
```

C specification:

```
#include "armpl.h"

void dpttrf_(const armpl_int_t *n, double *d, double *e, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry, the n diagonal elements of the tridiagonal matrix A. On exit, the n diagonal elements of the diagonal matrix D from the L*D*L^T factorization of A.

E Input and output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N-1). On entry, the (n-1) subdiagonal elements of the tridiagonal matrix A. On exit, the (n-1) subdiagonal elements of the unit bidiagonal factor L from the L*D*L^T factorization of A. E can also be regarded as the superdiagonal of the unit bidiagonal factor U from the U^T *D*U factorization of A.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value > 0: if INFO = k, the leading minor of order k is not positive definite; if k < N, the factorization could not be completed, while if k = N, the factorization was completed, but D(N) <= 0.

Related Information

For this routine in other precisions, please see *cpttrf*, *spttrf* and *zpttrf*. It also exists with a native C interface as *LAPACKE_dpttrf*.

4.3.88 dsptrf

`dsptrf` computes the factorization of a real symmetric matrix A stored in packed format using the Bunch-Kaufman diagonal pivoting method:

```
A = U*D*U**T  or  A = L*D*L**T
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsptrf(UPLO, N, AP, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void dsptrf_(const char *uplo, const armpl_int_t *n, double *ap,
             armpl_int_t *ipiv, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

`N` is `INTEGER`

The order of the matrix A. `N` >= 0.

AP Input and output parameter.

`AP` is `DOUBLE PRECISION`

`AP` is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array `AP` as follows: if `UPLO` = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if `UPLO` = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L, stored as a packed triangular matrix overwriting A (see below for further details).

IPIV Output parameter.

`IPIV` is `INTEGER` array, dimension (`N`)

Details of the interchanges and the block structure of D. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block. If `UPLO` = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If `UPLO` = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

INFO Output parameter.

`INFO` is `INTEGER`

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *csptrf*, *ssptrf* and *zsptrf*. It also exists with a native C interface as *LAPACKE_dsptrf*.

4.3.89 dsytrf

dsytrf computes the factorization of a real symmetric matrix A using the Bunch-Kaufman diagonal pivoting method. The form of the factorization is

```
A = U*D*U**T  or  A = L*D*L**T
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsytrf(UPLO, N, A, LDA, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dsytrf_(const char *uplo, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, armpl_int_t *ipiv, double *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L (see below for further details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >=1. For best performance LWORK >= N*NB, where NB is the block size returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *csytrf*, *ssytrf* and *zsytrf*. It also exists with a native C interface as *LAPACKE_dsytrf*.

4.3.90 dsytrf_aa

DSYTRF_AA computes the factorization of a real symmetric matrix A using the Aasen's algorithm. The form of the factorization is

```
A = U*T*U**T  or  A = L*T*L**T
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and T is a symmetric tridiagonal matrix.

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsytrf_aa(UPLO, N, A, LDA, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dsytrf_aa_(const char *uplo, const armpl_int_t *n, double *a,
                const armpl_int_t *lda, armpl_int_t *ipiv, double *work,
                const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, the tridiagonal matrix is stored in the diagonals and the subdiagonals of A just below (or above) the diagonals, and L is stored below (or above) the subdiaonals, when UPLO is 'L' (or 'U').

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

On exit, it contains the details of the interchanges, i.e., the row and column k of A were interchanged with the row and column IPIV(k).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= MAX(1,2*N). For optimum performance LWORK >= N*(1+NB), where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *csytrf_aa*, *ssytrf_aa* and *zsytrf_aa*. It also exists with a native C interface as *LAPACKE_dsytrf_aa*.

4.3.91 dsytrf_rk

DSYTRF_RK computes the factorization of a real symmetric matrix A using the bounded Bunch-Kaufman (rook) diagonal pivoting method:

```
A = P*U*D*(U**T)*(P**T) or A = P*L*D*(L**T)*(P**T),
```

where U (or L) is unit upper (or lower) triangular matrix, U^T (or L^T) is the transpose of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the blocked version of the algorithm, calling Level 3 BLAS. For more information see Further Details section.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsytrf_rk(UPLO, N, A, LDA, E, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dsytrf_rk_(const char *uplo, const armpl_int_t *n, double *a,
                const armpl_int_t *lda, double *e, armpl_int_t *ipiv,
                double *work, const armpl_int_t *lwork, armpl_int_t *info,
                ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U': the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced.

If UPLO = 'L': the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, contains: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N). On exit, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) is set to 0; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) is set to 0.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is set to 0 in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

IPIV describes the permutation matrix P in the factorization of matrix A as follows. The absolute value of IPIV(k) represents the index of row and column that were interchanged with the k-th row and column. The value of UPLO describes the order in which the interchanges were applied. Also, the sign of IPIV represents the block structure of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks which correspond to 1 or 2 interchanges at each factorization step. For more info see Further Details section.

If UPLO = 'U', (in factorization order, k decreases from N to 1): a) A single positive entry IPIV(k) > 0 means: D(k,k) is a 1-by-1 diagonal block. If IPIV(k) != k, rows and columns k and IPIV(k) were interchanged in the matrix A(1:N,1:N); If IPIV(k) = k, no interchange occurred.

b) A pair of consecutive negative entries IPIV(k) < 0 and IPIV(k-1) < 0 means: D(k-1:k,k-1:k) is a 2-by-2 diagonal block. (NOTE: negative entries in IPIV appear ONLY in pairs). 1) If -IPIV(k) != k, rows and columns k and -IPIV(k) were interchanged in the matrix A(1:N,1:N). If -IPIV(k) = k, no interchange occurred. 2) If -IPIV(k-1) != k-1, rows and columns k-1 and -IPIV(k-1) were interchanged in the matrix A(1:N,1:N). If -IPIV(k-1) = k-1, no interchange occurred.

c) In both cases a) and b), always ABS(IPIV(k)) <= k.

d) NOTE: Any entry IPIV(k) is always NONZERO on output.

If UPLO = 'L', (in factorization order, k increases from 1 to N): a) A single positive entry IPIV(k) > 0 means: D(k,k) is a 1-by-1 diagonal block. If IPIV(k) != k, rows and columns k and IPIV(k) were interchanged in the matrix A(1:N,1:N). If IPIV(k) = k, no interchange occurred.

b) A pair of consecutive negative entries IPIV(k) < 0 and IPIV(k+1) < 0 means: D(k:k+1,k:k+1) is a 2-by-2 diagonal block. (NOTE: negative entries in IPIV appear ONLY in pairs). 1) If -IPIV(k) != k, rows and columns k and -IPIV(k) were interchanged in the matrix A(1:N,1:N). If -IPIV(k) = k, no interchange occurred. 2) If -IPIV(k+1) != k+1, rows and columns k-1 and -IPIV(k-1) were interchanged in the matrix A(1:N,1:N). If -IPIV(k+1) = k+1, no interchange occurred.

c) In both cases a) and b), always ABS(IPIV(k)) >= k.

d) NOTE: Any entry IPIV(k) is always NONZERO on output.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)).. On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >=1. For best performance LWORK >= N*NB, where NB is the block size returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit

< 0: If INFO = -k, the k-th argument had an illegal value

> 0: If INFO = k, the matrix A is singular, because: If UPLO = 'U': column k in the upper triangular part of A contains all zeros. If UPLO = 'L': column k in the lower triangular part of A contains all zeros.

Therefore D(k,k) is exactly zero, and superdiagonal elements of column k of U (or subdiagonal elements of column k of L) are all zeros. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

NOTE: INFO only stores the first occurrence of a singularity, any subsequent occurrence of singularity is not stored in INFO even though the factorization always completes.

Related Information

For this routine in other precisions, please see *csytrf_rk*, *ssytrf_rk* and *zsytrf_rk*. It also exists with a native C interface as *LAPACKE_dsytrf_rk*.

4.3.92 dsytrf_rook

DSYTRF_ROOK computes the factorization of a real symmetric matrix A using the bounded Bunch-Kaufman ("rook") diagonal pivoting method. The form of the factorization is

```
A = U*D*U**T  or  A = L*D*L**T
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsytrf_rook(UPLO, N, A, LDA, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dsytrf_rook_(const char *uplo, const armpl_int_t *n, double *a,
                  const armpl_int_t *lda, armpl_int_t *ipiv, double *work,
                  const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L (see below for further details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D.

If UPLO = 'U': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k-1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k-1 and -IPIV(k-1) were inerchaged, D(k-1:k,k-1:k) is a 2-by-2 diagonal block.

If UPLO = 'L': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k+1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k+1 and -IPIV(k+1) were inerchaged, D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)).. On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >=1. For best performance LWORK >= N*NB, where NB is the block size returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *csytrf_rook*, *ssytrf_rook* and *zsytrf_rook*. It also exists with a native C interface as *LAPACKE_dsytrf_rook*.

4.3.93 dtplqt

dtplqt computes a blocked LQ factorization of a real "triangular-pentagonal" matrix C, which is composed of a triangular block A and pentagonal block B, using the compact WY representation for Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtplqt(M, N, L, MB, A, LDA, B, LDB, T, LDT, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dtplqt_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *l,
             const armpl_int_t *mb, double *a, const armpl_int_t *lda,
             double *b, const armpl_int_t *ldb, double *t,
             const armpl_int_t *ldt, double *work, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix B, and the order of the triangular matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix B. N >= 0.

L Input parameter.

L is INTEGER

The number of rows of the lower trapezoidal part of B. MIN(M, N) >= L >= 0. See Further Details.

MB Input parameter.

MB is INTEGER

The block size to be used in the blocked QR. M >= MB >= 1.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, M). On entry, the lower triangular M-by-M matrix A. On exit, the elements on and below the diagonal of the array contain the lower triangular matrix L.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). On entry, the pentagonal M-by-N matrix B. The first N-L columns are rectangular, and the last L columns are lower trapezoidal. On exit, B contains the pentagonal matrix V. See Further Details.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M).

T Output parameter.

T is DOUBLE PRECISION

T is an array, dimension (LDT, N). The lower triangular block reflectors stored in compact form as a sequence of upper triangular blocks. See Further Details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= MB.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MB*M) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctplqt*, *stplqt* and *ztplqt*.

4.3.94 dtpmlqt

DTPMQRT applies a real orthogonal matrix Q obtained from a "triangular-pentagonal" real block reflector H to a general real matrix C, which consists of two blocks A and B.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtpmlqt(SIDE, TRANS, M, N, K, L, MB, V, LDV, T, LDT, A, LDA, B,
                   LDB, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dtpmlqt_(const char *side, const char *trans, const armpl_int_t *m,
              const armpl_int_t *n, const armpl_int_t *k,
              const armpl_int_t *l, const armpl_int_t *mb, const double *v,
              const armpl_int_t *ldv, const double *t, const armpl_int_t *ldt,
              double *a, const armpl_int_t *lda, double *b,
              const armpl_int_t *ldb, double *work, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'T': Transpose, apply Q^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix B. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix B. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q.

L Input parameter.

L is INTEGER

The order of the trapezoidal part of V. K >= L >= 0. See Further Details.

MB Input parameter.

MB is INTEGER

The block size used for the storage of T. K >= MB >= 1. This must be the same value of MB used to generate T in DTPLQT.

V Input parameter.

V is DOUBLE PRECISION

V is an array, dimension (LDA, K). The i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,. . . ,k, as returned by DTPLQT in B. See Further Details.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. If SIDE = 'L', LDV >= max(1, M); if SIDE = 'R', LDV >= max(1, N).

T Input parameter.

T is DOUBLE PRECISION

T is an array, dimension (LDT, K). The upper triangular factors of the block reflectors as returned by DTPLQT, stored as a MB-by-K matrix.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= MB.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension. (LDA, N) if SIDE = 'L' or (LDA, K) if SIDE = 'R' On entry, the K-by-N or M-by-K matrix A. On exit, A is overwritten by the corresponding block of Q*C or Q^T *C or C*Q or C*Q^T . See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDC >= max(1, K); If SIDE = 'R', LDC >= max(1, M).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). On entry, the M-by-N matrix B. On exit, B is overwritten by the corresponding block of Q*C or Q^T *C or C*Q or C*Q^T . See Further Details.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M).

WORK Output parameter.

WORK is DOUBLE PRECISION array. The dimension of WORK is

N* MB if SIDE = 'L', or M* MB if SIDE = 'R'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctpmlqt*, *stpmlqt* and *ztpmlqt*.

4.3.95 dtpmqrt

dtpmqrt applies a real orthogonal matrix Q obtained from a "triangular-pentagonal" real block reflector H to a general real matrix C, which consists of two blocks A and B.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtpmqrt(SIDE, TRANS, M, N, K, L, NB, V, LDV, T, LDT, A, LDA, B,
                   LDB, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dtpmqrt_(const char *side, const char *trans, const armpl_int_t *m,
              const armpl_int_t *n, const armpl_int_t *k,
              const armpl_int_t *l, const armpl_int_t *nb, const double *v,
              const armpl_int_t *ldv, const double *t, const armpl_int_t *ldt,
              double *a, const armpl_int_t *lda, double *b,
              const armpl_int_t *ldb, double *work, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'T': Transpose, apply Q^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix B. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix B. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q.

L Input parameter.

L is INTEGER

The order of the trapezoidal part of V. K >= L >= 0. See Further Details.

NB Input parameter.

NB is INTEGER

The block size used for the storage of T. K >= NB >= 1. This must be the same value of NB used to generate T in CTPQRT.

V Input parameter.

V is DOUBLE PRECISION

V is an array, dimension (LDA, K). The i-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,. . . ,k, as returned by CTPQRT in B. See Further Details.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. If SIDE = 'L', LDV >= max(1, M); if SIDE = 'R', LDV >= max(1, N).

T Input parameter.

T is DOUBLE PRECISION

T is an array, dimension (LDT, K). The upper triangular factors of the block reflectors as returned by CTPQRT, stored as a NB-by-K matrix.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= NB.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension. (LDA, N) if SIDE = 'L' or (LDA, K) if SIDE = 'R' On entry, the K-by-N or M-by-K matrix A. On exit, A is overwritten by the corresponding block of Q*C or Q^T *C or C*Q or C*Q^T . See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDC >= max(1, K); If SIDE = 'R', LDC >= max(1, M).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). On entry, the M-by-N matrix B. On exit, B is overwritten by the corresponding block of Q*C or Q^T *C or C*Q or C*Q^T . See Further Details.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M).

WORK Output parameter.

WORK is DOUBLE PRECISION array. The dimension of WORK is

N* NB if SIDE = 'L', or M* NB if SIDE = 'R'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctpmqrt*, *stpmqrt* and *ztpmqrt*. It also exists with a native C interface as *LAPACKE_dtpmqrt*.

4.3.96 dtpqrt

dtpqrt computes a blocked QR factorization of a real "triangular-pentagonal" matrix C, which is composed of a triangular block A and pentagonal block B, using the compact WY representation for Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtpqrt(M, N, L, NB, A, LDA, B, LDB, T, LDT, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dtpqrt_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *l,
             const armpl_int_t *nb, double *a, const armpl_int_t *lda,
             double *b, const armpl_int_t *ldb, double *t,
             const armpl_int_t *ldt, double *work, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix B. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix B, and the order of the triangular matrix A. N >= 0.

L Input parameter.

L is INTEGER

The number of rows of the upper trapezoidal part of B. MIN(M, N) >= L >= 0. See Further Details.

NB Input parameter.

NB is INTEGER

The block size to be used in the blocked QR. N >= NB >= 1.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the upper triangular N-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the upper triangular matrix R.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). On entry, the pentagonal M-by-N matrix B. The first M-L rows are rectangular, and the last L rows are upper trapezoidal. On exit, B contains the pentagonal matrix V. See Further Details.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M).

T Output parameter.

T is DOUBLE PRECISION

T is an array, dimension (LDT, N). The upper triangular block reflectors stored in compact form as a sequence of upper triangular blocks. See Further Details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= NB.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (NB*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctpqrt*, *stpqrt* and *ztpqrt*. It also exists with a native C interface as *LAPACKE_dtpqrt*.

4.3.97 dtzrzf

dtzrzf reduces the M-by-N (M<=N) real upper trapezoidal matrix A to upper triangular form by means of orthogonal transformations.

The upper trapezoidal matrix A is factored as

```
A = ( R  0 ) * Z,
```

where Z is an N-by-N orthogonal matrix and R is an M-by-M upper triangular matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtzrzf(M, N, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dtzrzf_(const armpl_int_t *m, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, double *tau, double *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= M.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the leading M-by-N upper trapezoidal part of the array A must contain the matrix to be factorized. On exit, the leading M-by-M upper triangular part of A contains the upper triangular matrix R, and elements M+1 to N of the first M rows of A, with the array TAU, represent the orthogonal matrix Z as a product of M elementary reflectors.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (M). The scalar factors of the elementary reflectors.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, M). For optimum performance LWORK >= M*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctzrzf*, *stzrzf* and *ztzrzf*. It also exists with a native C interface as *LAPACKE_dtzrzf*.

4.3.98 sgbtrf

sgbtrf computes an LU factorization of a real m-by-n band matrix A using partial pivoting with row interchanges.

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgbtrf(M, N, KL, KU, AB, LDAB, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void sgbtrf_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *kl, const armpl_int_t *ku, float *ab,
             const armpl_int_t *ldab, armpl_int_t *ipiv, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

AB Input and output parameter.

AB is REAL

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows KL+1 to 2*KL+KU+1; rows 1 to KL of the array need not be set. The j-th column of A is stored in the j-th column of the array AB as follows: AB(kl+ku+1+i-j,j) = A(i,j) for max(1,j-ku)<=i<=min(m,j+kl)

On exit, details of the factorization: U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1. See below for further details.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= 2*KL+KU+1.

IPIV Output parameter.

IPIV is INTEGER array, dimension (min(M, N))

The pivot indices; for 1 <= i <= min(M, N), row i of the matrix was interchanged with row IPIV(i).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = +i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *cgbtrf*, *dgbtrf* and *zgbtrf*. It also exists with a native C interface as *LAPACKE_sgbtrf*.

4.3.99 sgelq

sgelq computes a LQ factorization of an M-by-N matrix A.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgelq(M, N, A, LDA, T, TSIZE, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgelq_(const armpl_int_t *m, const armpl_int_t *n, float *a,
            const armpl_int_t *lda, float *t, const armpl_int_t *tsize,
            float *work, const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and below the diagonal of the array contain the M-by-min(M, N) lower trapezoidal matrix L (L is lower triangular if M <= N); the elements above the diagonal are used to store part of the data structure to represent Q.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

T Output parameter.

T is REAL

T is an array, dimension (MAX(5, TSIZE)). On exit, if INFO = 0, T(1) returns optimal (or either minimal or optimal, if query is assumed) TSIZE. See TSIZE for details. Remaining T contains part of the data structure used to represent Q. If one wants to apply or construct Q, then one needs to keep T (in addition to A) and pass it to further subroutines.

TSIZE Input parameter.

TSIZE is INTEGER

If TSIZE >= 5, the dimension of the array T. If TSIZE = -1 or -2, then a workspace query is assumed. The routine only calculates the sizes of the T and WORK arrays, returns these values as the first entries of the T and WORK arrays, and no error message related to T or WORK is issued by XERBLA. If TSIZE = -1, the routine calculates optimal size of T for the optimum performance and returns this value in T(1). If TSIZE = -2, the routine calculates minimal size of T and returns this value in T(1).

WORK Output parameter.

(workspace) REAL

(workspace) is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) contains optimal (or either minimal or optimal, if query was assumed) LWORK. See LWORK for details.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If LWORK = -1 or -2, then a workspace query is assumed. The routine only calculates the sizes of the T and WORK arrays, returns these values as the first entries of the T and WORK arrays, and no error message related to T or WORK is issued by XERBLA. If LWORK = -1, the routine calculates optimal size of WORK for the optimal performance and returns this value in WORK(1). If LWORK = -2, the routine calculates minimal size of WORK and returns this value in WORK(1).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgelq*, *dgelq* and *zgelq*. It also exists with a native C interface as *LAPACKE_sgelq*.

4.3.100 sgelqf

sgelqf computes an LQ factorization of a real M-by-N matrix A: A = L * Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgelqf(M, N, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgelqf_(const armpl_int_t *m, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, float *tau, float *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and below the diagonal of the array contain the m-by-min(m,n) lower trapezoidal matrix L (L is lower triangular if m <= n); the elements above the diagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is REAL

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, M). For optimum performance LWORK >= M*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgelqf*, *dgelqf* and *zgelqf*. It also exists with a native C interface as *LAPACKE_sgelqf*.

4.3.101 sgelqt

DGELQT computes a blocked LQ factorization of a real M-by-N matrix A using the compact WY representation of Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgelqt(M, N, MB, A, LDA, T, LDT, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgelqt_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *mb, float *a, const armpl_int_t *lda,
             float *t, const armpl_int_t *ldt, float *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

MB Input parameter.

MB is INTEGER

The block size to be used in the blocked QR. MIN(M, N) >= MB >= 1.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and below the diagonal of the array contain the M-by-MIN(M, N) lower trapezoidal matrix L (L is lower triangular if M <= N); the elements above the diagonal are the rows of V.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

T Output parameter.

T is REAL

T is an array, dimension (LDT,MIN(M, N)). The upper triangular block reflectors stored in compact form as a sequence of upper triangular blocks. See below for further details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= MB.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MB*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgelqt*, *dgelqt* and *zgelqt*.

4.3.102 sgemlq

sgemlq overwrites the general real M-by-N matrix C with SIDE = 'L' SIDE = 'R' TRANS = 'N': Q * C C * Q TRANS = 'T': Q^T * C C * Q^T where Q is a real orthogonal matrix defined as the product of blocked elementary reflectors computed by short wide LQ factorization (SGELQ)

Syntax

Fortran specification:

```
use armpl_library

subroutine sgemlq(SIDE, TRANS, M, N, K, A, LDA, T, TSIZE, C, LDC, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void sgemlq_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k, const float *a,
             const armpl_int_t *lda, const float *t, const armpl_int_t *tsize,
             float *c, const armpl_int_t *ldc, float *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'T': Transpose, apply Q^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >=0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is REAL

A is an array, dimension. (LDA, M) if SIDE = 'L', (LDA, N) if SIDE = 'R' Part of the data structure to represent Q as returned by DGELQ.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, K).

T Input parameter.

T is REAL

T is an array, dimension (MAX(5, TSIZE)).. Part of the data structure to represent Q as returned by SGELQ.

TSIZE Input parameter.

TSIZE is INTEGER

The dimension of the array T. TSIZE >= 5.

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

(workspace) REAL

(workspace) is an array, dimension (MAX(1, LWORK)) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If LWORK = -1, then a workspace query is assumed. The routine only calculates the size of the WORK array, returns this value as WORK(1), and no error message related to WORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgemlq*, *dgemlq* and *zgemlq*. It also exists with a native C interface as *LAPACKE_sgemlq*.

4.3.103 sgemlqt

DGEMLQT overwrites the general real M-by-N matrix C with SIDE = 'L' SIDE = 'R' TRANS = 'N': Q C C Q TRANS = 'T': Q^T C C Q^T where Q is a real orthogonal matrix defined as the product of K elementary reflectors: Q = H(1) H(2) . . . H(K) = I - V T V^T generated using the compact WY representation as returned by DGELQT. Q is of order M if SIDE = 'L' and of order N if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgemlqt(SIDE, TRANS, M, N, K, MB, V, LDV, T, LDT, C, LDC, WORK,
                   INFO)
```

C specification:

```
#include "armpl.h"

void sgemlqt_(const char *side, const char *trans, const armpl_int_t *m,
              const armpl_int_t *n, const armpl_int_t *k,
              const armpl_int_t *mb, const float *v, const armpl_int_t *ldv,
              const float *t, const armpl_int_t *ldt, float *c,
              const armpl_int_t *ldc, float *work, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'C': Transpose, apply Q^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

MB Input parameter.

MB is INTEGER

The block size used for the storage of T. K >= MB >= 1. This must be the same value of MB used to generate T in DGELQT.

V Input parameter.

V is REAL

V is an array, dimension. (LDV, M) if SIDE = 'L', (LDV, N) if SIDE = 'R' The i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by DGELQT in the first K rows of its array argument A.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. LDV >= max(1, K).

T Input parameter.

T is REAL

T is an array, dimension (LDT, K). The upper triangular factors of the block reflectors as returned by DGELQT, stored as a MB-by-K matrix.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= MB.

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q C, Q^T C, C Q^T or C Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is REAL array. The dimension of

WORK is N* MB if SIDE = 'L', or M* MB if SIDE = 'R'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgemlqt*, *dgemlqt* and *zgemlqt*.

4.3.104 sgemqr

sgemqr overwrites the general real M-by-N matrix C with SIDE = 'L' SIDE = 'R' TRANS = 'N': Q * C C * Q TRANS = 'T': Q^T * C C * Q^T where Q is a real orthogonal matrix defined as the product of blocked elementary reflectors computed by tall skinny QR factorization (SGEQR)

Syntax

Fortran specification:

```
use armpl_library

subroutine sgemqr(SIDE, TRANS, M, N, K, A, LDA, T, TSIZE, C, LDC, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void sgemqr_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k, const float *a,
             const armpl_int_t *lda, const float *t, const armpl_int_t *tsize,
             float *c, const armpl_int_t *ldc, float *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'T': Transpose, apply Q^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >=0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, K). Part of the data structure to represent Q as returned by SGEQR.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDA >= max(1, M); if SIDE = 'R', LDA >= max(1, N).

T Input parameter.

T is REAL

T is an array, dimension (MAX(5, TSIZE)).. Part of the data structure to represent Q as returned by SGEQR.

TSIZE Input parameter.

TSIZE is INTEGER

The dimension of the array T. TSIZE >= 5.

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

(workspace) REAL

(workspace) is an array, dimension (MAX(1, LWORK)) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If LWORK = -1, then a workspace query is assumed. The routine only calculates the size of the WORK array, returns this value as WORK(1), and no error message related to WORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgemqr*, *dgemqr* and *zgemqr*. It also exists with a native C interface as *LAPACKE_sgemqr*.

4.3.105 sgemqrt

sgemqrt overwrites the general real M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': Q C C Q TRANS = 'T': Q^T C C Q^T

where Q is a real orthogonal matrix defined as the product of K elementary reflectors:

```
Q = H(1) H(2) . . . H(K) = I - V T V**T
```

generated using the compact WY representation as returned by SGEQRT.

Q is of order M if SIDE = 'L' and of order N if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgemqrt(SIDE, TRANS, M, N, K, NB, V, LDV, T, LDT, C, LDC, WORK,
                   INFO)
```

C specification:

```
#include "armpl.h"

void sgemqrt_(const char *side, const char *trans, const armpl_int_t *m,
              const armpl_int_t *n, const armpl_int_t *k,
              const armpl_int_t *nb, const float *v, const armpl_int_t *ldv,
              const float *t, const armpl_int_t *ldt, float *c,
              const armpl_int_t *ldc, float *work, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'T': Transpose, apply Q^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

NB Input parameter.

NB is INTEGER

The block size used for the storage of T. K >= NB >= 1. This must be the same value of NB used to generate T in CGEQRT.

V Input parameter.

V is REAL

V is an array, dimension (LDV, K). The i-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,. . . ,k, as returned by CGEQRT in the first K columns of its array argument A.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. If SIDE = 'L', LDA >= max(1, M); if SIDE = 'R', LDA >= max(1, N).

T Input parameter.

T is REAL

T is an array, dimension (LDT, K). The upper triangular factors of the block reflectors as returned by CGEQRT, stored as a NB-by-N matrix.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= NB.

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q C, Q^T C, C Q^T or C Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is REAL array. The dimension of WORK is

N* NB if SIDE = 'L', or M* NB if SIDE = 'R'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgemqrt*, *dgemqrt* and *zgemqrt*. It also exists with a native C interface as *LAPACKE_sgemqrt*.

4.3.106 sgeqlf

sgeqlf computes a QL factorization of a real M-by-N matrix A: A = Q * L.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgeqlf(M, N, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgeqlf_(const armpl_int_t *m, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, float *tau, float *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, if m >= n, the lower triangle of the subarray A(m-n+1:m,1:n) contains the N-by-N lower triangular matrix L; if m <= n, the elements on and below the (n-m)-th superdiagonal contain the M-by-N lower trapezoidal matrix L; the remaining elements, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is REAL

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N). For optimum performance LWORK >= N*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgeqlf*, *dgeqlf* and *zgeqlf*. It also exists with a native C interface as *LAPACKE_sgeqlf*.

4.3.107 sgeqp3

`sgeqp3` computes a QR factorization with column pivoting of a matrix A: A*P = Q*R using Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgeqp3(M, N, A, LDA, JPVT, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgeqp3_(const armpl_int_t *m, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, armpl_int_t *jpvt, float *tau,
             float *work, const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

`M` is `INTEGER`

The number of rows of the matrix `A`. `M` >= 0.

N Input parameter.

`N` is `INTEGER`

The number of columns of the matrix `A`. `N` >= 0.

A Input and output parameter.

`A` is `REAL`

`A` is an array, dimension (`LDA`, `N`). On entry, the M-by-N matrix `A`. On exit, the upper triangle of the array contains the min(`M`, `N`)-by-N upper trapezoidal matrix R; the elements below the diagonal, together with the array `TAU`, represent the orthogonal matrix Q as a product of min(`M`, `N`) elementary reflectors.

LDA Input parameter.

`LDA` is `INTEGER`

The leading dimension of the array `A`. `LDA` >= max(1, `M`).

JPVT Input and output parameter.

`JPVT` is `INTEGER` array, dimension (`N`)

On entry, if JPVT(J).ne.0, the J-th column of `A` is permuted to the front of `A`*P (a leading column); if JPVT(J)=0, the J-th column of `A` is a free column. On exit, if JPVT(J)=K, then the J-th column of `A`*P was the the K-th column of `A`.

TAU Output parameter.

TAU is REAL

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO=0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= 3*N+1. For optimal performance LWORK >= 2*N+(N+1)*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cgeqp3*, *dgeqp3* and *zgeqp3*. It also exists with a native C interface as *LAPACKE_sgeqp3*.

4.3.108 sgeqr

sgeqr computes a QR factorization of an M-by-N matrix A.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgeqr(M, N, A, LDA, T, TSIZE, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgeqr_(const armpl_int_t *m, const armpl_int_t *n, float *a,
            const armpl_int_t *lda, float *t, const armpl_int_t *tsize,
            float *work, const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the min(M, N)-by-N upper trapezoidal matrix R (R is upper triangular if M >= N); the elements below the diagonal are used to store part of the data structure to represent Q.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

T Output parameter.

T is REAL

T is an array, dimension (MAX(5, TSIZE)). On exit, if INFO = 0, T(1) returns optimal (or either minimal or optimal, if query is assumed) TSIZE. See TSIZE for details. Remaining T contains part of the data structure used to represent Q. If one wants to apply or construct Q, then one needs to keep T (in addition to A) and pass it to further subroutines.

TSIZE Input parameter.

TSIZE is INTEGER

If TSIZE >= 5, the dimension of the array T. If TSIZE = -1 or -2, then a workspace query is assumed. The routine only calculates the sizes of the T and WORK arrays, returns these values as the first entries of the T and WORK arrays, and no error message related to T or WORK is issued by XERBLA. If TSIZE = -1, the routine calculates optimal size of T for the optimum performance and returns this value in T(1). If TSIZE = -2, the routine calculates minimal size of T and returns this value in T(1).

WORK Output parameter.

(workspace) REAL

(workspace) is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) contains optimal (or either minimal or optimal, if query was assumed) LWORK. See LWORK for details.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If LWORK = -1 or -2, then a workspace query is assumed. The routine only calculates the sizes of the T and WORK arrays, returns these values as the first entries of the T and WORK arrays, and no error message related to T or WORK is issued by XERBLA. If LWORK = -1, the routine calculates optimal size of WORK for the optimal performance and returns this value in WORK(1). If LWORK = -2, the routine calculates minimal size of WORK and returns this value in WORK(1).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgeqr*, *dgeqr* and *zgeqr*. It also exists with a native C interface as *LAPACKE_sgeqr*.

4.3.109 sgeqrf

`sgeqrf` computes a QR factorization of a real M-by-N matrix A: A = Q * R.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgeqrf(M, N, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgeqrf_(const armpl_int_t *m, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, float *tau, float *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the min(M, N)-by-N upper trapezoidal matrix R (R is upper triangular if m >= n); the elements below the diagonal, with the array TAU, represent the orthogonal matrix Q as a product of min(m,n) elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is REAL

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N). For optimum performance LWORK >= N*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgeqrf*, *dgeqrf* and *zgeqrf*. It also exists with a native C interface as *LAPACKE_sgeqrf*.

4.3.110 sgeqrfp

sgeqrfp computes a QR factorization of a real M-by-N matrix A: A = Q * R. The diagonal entries of R are nonnegative.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgeqrfp(M, N, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgeqrfp_(const armpl_int_t *m, const armpl_int_t *n, float *a,
              const armpl_int_t *lda, float *tau, float *work,
              const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the min(M, N)-by-N upper trapezoidal matrix R (R is upper triangular if m >= n). The diagonal entries of R are nonnegative; the elements below the diagonal, with the array TAU, represent the orthogonal matrix Q as a product of min(m,n) elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is REAL

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N). For optimum performance LWORK >= N*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgeqrfp*, *dgeqrfp* and *zgeqrfp*. It also exists with a native C interface as *LAPACKE_sgeqrfp*.

4.3.111 sgeqrt

sgeqrt computes a blocked QR factorization of a real M-by-N matrix A using the compact WY representation of Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgeqrt(M, N, NB, A, LDA, T, LDT, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgeqrt_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *nb, float *a, const armpl_int_t *lda,
             float *t, const armpl_int_t *ldt, float *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

NB Input parameter.

NB is INTEGER

The block size to be used in the blocked QR. MIN(M, N) >= NB >= 1.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the min(M, N)-by-N upper trapezoidal matrix R (R is upper triangular if M >= N); the elements below the diagonal are the columns of V.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

T Output parameter.

T is REAL

T is an array, dimension (LDT,MIN(M, N)). The upper triangular block reflectors stored in compact form as a sequence of upper triangular blocks. See below for further details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= NB.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (NB*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgeqrt*, *dgeqrt* and *zgeqrt*. It also exists with a native C interface as *LAPACKE_sgeqrt*.

4.3.112 sgerqf

sgerqf computes an RQ factorization of a real M-by-N matrix A: A = R * Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgerqf(M, N, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgerqf_(const armpl_int_t *m, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, float *tau, float *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, if m <= n, the upper triangle of the subarray A(1:m,n-m+1:n) contains the M-by-M upper triangular matrix R; if m >= n, the elements on and above the (m-n)-th subdiagonal contain the M-by-N upper trapezoidal matrix R; the remaining elements, with the array TAU, represent the orthogonal matrix Q as a product of min(m,n) elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is REAL

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, M). For optimum performance LWORK >= M*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgerqf*, *dgerqf* and *zgerqf*. It also exists with a native C interface as *LAPACKE_sgerqf*.

4.3.113 sgetrf

sgetrf computes an LU factorization of a general M-by-N matrix A using partial pivoting with row interchanges.

The factorization has the form

```
A = P * L * U
```

where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if m > n), and U is upper triangular (upper trapezoidal if m < n).

This is the right-looking Level 3 BLAS version of the algorithm.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgetrf(M, N, A, LDA, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void sgetrf_(const armpl_int_t *m, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, armpl_int_t *ipiv, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the M-by-N matrix to be factored. On exit, the factors L and U from the factorization A = P*L*U; the unit diagonal elements of L are not stored.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

IPIV Output parameter.

IPIV is INTEGER array, dimension (min(M, N))

The pivot indices; for 1 <= i <= min(M, N), row i of the matrix was interchanged with row IPIV(i).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *cgetrf*, *dgetrf* and *zgetrf*. It also exists with a native C interface as *LAPACKE_sgetrf*.

4.3.114 sgetrf2

sgetrf2 computes an LU factorization of a general M-by-N matrix A using partial pivoting with row interchanges.

The factorization has the form

```
A = P * L * U
```

where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if m > n), and U is upper triangular (upper trapezoidal if m < n).

This is the recursive version of the algorithm. It divides the matrix into four submatrices:

```
    [  A11 | A12  ]  where A11 is n1 by n1 and A22 is n2 by n2
A = [ -----|-----  ]  with n1 = min(m,n)/2
    [  A21 | A22  ]       n2 = n-n1

                                  [ A11 ]
```

The subroutine calls itself to factor [—],

```
                      [ A12 ]
[ A12 ]
```

do the swaps on [—], solve A12, update A22,

```
[ A22 ]
```

then calls itself to factor A22 and do the swaps on A21.

Syntax

Fortran specification:

```
use armpl_library

recursive subroutine sgetrf2(M, N, A, LDA, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void sgetrf2_(const armpl_int_t *m, const armpl_int_t *n, float *a,
              const armpl_int_t *lda, armpl_int_t *ipiv, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the M-by-N matrix to be factored. On exit, the factors L and U from the factorization A = P*L*U; the unit diagonal elements of L are not stored.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

IPIV Output parameter.

IPIV is INTEGER array, dimension (min(M, N))

The pivot indices; for 1 <= i <= min(M, N), row i of the matrix was interchanged with row IPIV(i).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *cgetrf2*, *dgetrf2* and *zgetrf2*. It also exists with a native C interface as *LAPACKE_sgetrf2*.

4.3.115 sggqrf

sggqrf computes a generalized QR factorization of an N-by-M matrix A and an N-by-P matrix B:

```
A = Q*R,        B = Q*T*Z,
```

where Q is an N-by-N orthogonal matrix, Z is a P-by-P orthogonal matrix, and R and T assume one of the forms:

if N >= M, R = (R11) M , or if N < M, R = (R11 R12) N,

```
(  0  ) N-M                          N   M-N
  M
```

where R11 is upper triangular, and

if N <= P, T = (0 T12) N, or if N > P, T = (T11) N-P,

```
P-N  N                              ( T21 ) P
                                      P
```

where T12 or T21 is upper triangular.

In particular, if B is square and nonsingular, the GQR factorization of A and B implicitly gives the QR factorization of inv(B)*A:

```
inv(B)*A = Z**T*(inv(T)*R)
```

where inv(B) denotes the inverse of the matrix B, and Z^T denotes the transpose of the matrix Z.

Syntax

Fortran specification:

```
use armpl_library

subroutine sggqrf(N, M, P, A, LDA, TAUA, B, LDB, TAUB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sggqrf_(const armpl_int_t *n, const armpl_int_t *m, const armpl_int_t *p,
             float *a, const armpl_int_t *lda, float *taua, float *b,
             const armpl_int_t *ldb, float *taub, float *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The number of rows of the matrices A and B. N >= 0.

M Input parameter.

M is INTEGER

The number of columns of the matrix A. M >= 0.

P Input parameter.

P is INTEGER

The number of columns of the matrix B. P >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, M). On entry, the N-by-M matrix A. On exit, the elements on and above the diagonal of the array contain the min(N, M)-by-M upper trapezoidal matrix R (R is upper triangular if N >= M); the elements below the diagonal, with the array TAUA, represent the orthogonal matrix Q as a product of min(N, M) elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TAUA Output parameter.

TAUA is REAL

TAUA is an array, dimension (min(N, M)). The scalar factors of the elementary reflectors which represent the orthogonal matrix Q (see Further Details).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, P). On entry, the N-by-P matrix B. On exit, if N <= P, the upper triangle of the subarray B(1:N,P-N+1:P) contains the N-by-N upper triangular matrix T; if N > P, the elements on and above the (N-P)-th subdiagonal contain the N-by-P upper trapezoidal matrix T; the remaining elements, with the array TAUB, represent the orthogonal matrix Z as a product of elementary reflectors (see Further Details).

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

TAUB Output parameter.

TAUB is REAL

TAUB is an array, dimension (min(N, P)). The scalar factors of the elementary reflectors which represent the orthogonal matrix Z (see Further Details).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N, M, P). For optimum performance LWORK >= max(N, M, P)*max(NB1,NB2,NB3), where NB1 is the optimal blocksize for the QR factorization of an N-by-M matrix, NB2 is the optimal blocksize for the RQ factorization of an N-by-P matrix, and NB3 is the optimal blocksize for a call of SORMQR.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cggqrf*, *dggqrf* and *zggqrf*. It also exists with a native C interface as *LAPACKE_sggqrf*.

4.3.116 sggrqf

sggrqf computes a generalized RQ factorization of an M-by-N matrix A and a P-by-N matrix B:

```
A = R*Q,        B = Z*T*Q,
```

where Q is an N-by-N orthogonal matrix, Z is a P-by-P orthogonal matrix, and R and T assume one of the forms:

if M <= N, R = (0 R12) M, or if M > N, R = (R11) M-N,

```
N-M  M                              ( R21 ) N
                                      N
```

where R12 or R21 is upper triangular, and

if P >= N, T = (T11) N , or if P < N, T = (T11 T12) P,

```
(  0  ) P-N                          P   N-P
   N
```

where T11 is upper triangular.

In particular, if B is square and nonsingular, the GRQ factorization of A and B implicitly gives the RQ factorization of A*inv(B):

```
A*inv(B) = (R*inv(T))*Z**T
```

where inv(B) denotes the inverse of the matrix B, and Z^T denotes the transpose of the matrix Z.

Syntax

Fortran specification:

```
use armpl_library

subroutine sggrqf(M, P, N, A, LDA, TAUA, B, LDB, TAUB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sggrqf_(const armpl_int_t *m, const armpl_int_t *p, const armpl_int_t *n,
             float *a, const armpl_int_t *lda, float *taua, float *b,
             const armpl_int_t *ldb, float *taub, float *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

P Input parameter.

P is INTEGER

The number of rows of the matrix B. P >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrices A and B. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, if M <= N, the upper triangle of the subarray A(1:M,N-M+1:N) contains the M-by-M upper triangular matrix R; if M > N, the elements on and above the (M-N)-th subdiagonal contain the M-by-N upper trapezoidal matrix R; the remaining elements, with the array TAUA, represent the orthogonal matrix Q as a product of elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAUA Output parameter.

TAUA is REAL

TAUA is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors which represent the orthogonal matrix Q (see Further Details).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, N). On entry, the P-by-N matrix B. On exit, the elements on and above the diagonal of the array contain the min(P, N)-by-N upper trapezoidal matrix T (T is upper triangular if P >= N); the elements below the diagonal, with the array TAUB, represent the orthogonal matrix Z as a product of elementary reflectors (see Further Details).

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, P).

TAUB Output parameter.

TAUB is REAL

TAUB is an array, dimension (min(P, N)). The scalar factors of the elementary reflectors which represent the orthogonal matrix Z (see Further Details).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N, M, P). For optimum performance LWORK >= max(N, M, P)*max(NB1,NB2,NB3), where NB1 is the optimal blocksize for the RQ factorization of an M-by-N matrix, NB2 is the optimal blocksize for the QR factorization of a P-by-N matrix, and NB3 is the optimal blocksize for a call of SORMRQ.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INF0= -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cggrqf*, *dggrqf* and *zggrqf*. It also exists with a native C interface as *LAPACKE_sggrqf*.

4.3.117 sgttrf

`sgttrf` computes an LU factorization of a real tridiagonal matrix A using elimination with partial pivoting and row interchanges.

The factorization has the form

```
A = L * U
```

where L is a product of permutation and unit lower bidiagonal matrices and U is upper triangular with nonzeros in only the main diagonal and first two superdiagonals.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgttrf(N, DL, D, DU, DU2, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void sgttrf_(const armpl_int_t *n, float *dl, float *d, float *du, float *du2,
             armpl_int_t *ipiv, armpl_int_t *info);
```

Parameters

N Input parameter.

`N` is `INTEGER`

The order of the matrix A.

DL Input and output parameter.

`DL` is `REAL`

`DL` is an array, dimension (N-1). On entry, `DL` must contain the (n-1) sub-diagonal elements of A.

On exit, `DL` is overwritten by the (n-1) multipliers that define the matrix L from the LU factorization of A.

D Input and output parameter.

`D` is `REAL`

`D` is an array, dimension (`N`). On entry, `D` must contain the diagonal elements of A.

On exit, `D` is overwritten by the n diagonal elements of the upper triangular matrix U from the LU factorization of A.

DU Input and output parameter.

`DU` is `REAL`

`DU` is an array, dimension (N-1). On entry, `DU` must contain the (n-1) super-diagonal elements of A.

On exit, `DU` is overwritten by the (n-1) elements of the first super-diagonal of U.

DU2 Output parameter.

DU2 is REAL

DU2 is an array, dimension (N-2). On exit, DU2 is overwritten by the (n-2) elements of the second super-diagonal of U.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices; for 1 <= i <= n, row i of the matrix was interchanged with row IPIV(i). IPIV(i) will always be either i or i+1; IPIV(i) = i indicates a row interchange was not required.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value > 0: if INFO = k, U(k,k) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *cgttrf*, *dgttrf* and *zgttrf*. It also exists with a native C interface as *LAPACKE_sgttrf*.

4.3.118 sorglq

sorglq generates an M-by-N real matrix Q with orthonormal rows, which is defined as the first M rows of a product of K elementary reflectors of order N

```
Q  =  H(k) . . . H(2) H(1)
```

as returned by SGELQF.

Syntax

Fortran specification:

```
use armpl_library

subroutine sorglq(M, N, K, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sorglq_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *k,
             float *a, const armpl_int_t *lda, const float *tau, float *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix Q. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix Q. N >= M.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. M >= K >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by SGELQF in the first k rows of its array argument A. On exit, the M-by-N matrix Q.

LDA Input parameter.

LDA is INTEGER

The first dimension of the array A. LDA >= max(1, M).

TAU Input parameter.

TAU is REAL

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by SGELQF.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, M). For optimum performance LWORK >= M*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument has an illegal value

Related Information

For this routine in other precisions, please see *dorglq*. It also exists with a native C interface as *LAPACKE_sorglq*.

4.3.119 sorgql

sorgql generates an M-by-N real matrix Q with orthonormal columns, which is defined as the last N columns of a product of K elementary reflectors of order M

```
Q  =  H(k) . . . H(2) H(1)
```

as returned by SGEQLF.

Syntax

Fortran specification:

```
use armpl_library

subroutine sorgql(M, N, K, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sorgql_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *k,
             float *a, const armpl_int_t *lda, const float *tau, float *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix Q. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix Q. M >= N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. N >= K >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the (n-k+i)-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by SGEQLF in the last k columns of its array argument A. On exit, the M-by-N matrix Q.

LDA Input parameter.

LDA is INTEGER

The first dimension of the array A. LDA >= max(1, M).

TAU Input parameter.

TAU is REAL

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by SGEQLF.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N). For optimum performance LWORK >= N*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument has an illegal value

Related Information

For this routine in other precisions, please see *dorgql*. It also exists with a native C interface as *LAPACKE_sorgql*.

4.3.120 sorgqr

sorgqr generates an M-by-N real matrix Q with orthonormal columns, which is defined as the first N columns of a product of K elementary reflectors of order M

```
Q  =  H(1) H(2) . . . H(k)
```

as returned by SGEQRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine sorgqr(M, N, K, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sorgqr_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *k,
             float *a, const armpl_int_t *lda, const float *tau, float *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix Q. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix Q. M >= N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. N >= K >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the i-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,. . . ,k, as returned by SGEQRF in the first k columns of its array argument A. On exit, the M-by-N matrix Q.

LDA Input parameter.

LDA is INTEGER

The first dimension of the array A. LDA >= max(1, M).

TAU Input parameter.

TAU is REAL

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by SGEQRF.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N). For optimum performance LWORK >= N*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument has an illegal value

Related Information

For this routine in other precisions, please see *dorgqr*. It also exists with a native C interface as *LAPACKE_sorgqr*.

4.3.121 sorgrq

sorgrq generates an M-by-N real matrix Q with orthonormal rows, which is defined as the last M rows of a product of K elementary reflectors of order N

```
Q  =  H(1) H(2) . . . H(k)
```

as returned by SGERQF.

Syntax

Fortran specification:

```
use armpl_library

subroutine sorgrq(M, N, K, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sorgrq_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *k,
             float *a, const armpl_int_t *lda, const float *tau, float *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix Q. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix Q. N >= M.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. M >= K >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the (m-k+i)-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by SGERQF in the last k rows of its array argument A. On exit, the M-by-N matrix Q.

LDA Input parameter.

LDA is INTEGER

The first dimension of the array A. LDA >= max(1, M).

TAU Input parameter.

TAU is REAL

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by SGERQF.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, M). For optimum performance LWORK >= M*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument has an illegal value

Related Information

For this routine in other precisions, please see *dorgrq*. It also exists with a native C interface as *LAPACKE_sorgrq*.

4.3.122 sormlq

sormlq overwrites the general real M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'T': Q^T * C C * Q^T

where Q is a real orthogonal matrix defined as the product of k elementary reflectors

```
Q = H(k) . . . H(2) H(1)
```

as returned by SGELQF. Q is of order M if SIDE = 'L' and of order N if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine sormlq(SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void sormlq_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k, const float *a,
             const armpl_int_t *lda, const float *tau, float *c,
             const armpl_int_t *ldc, float *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'T': Transpose, apply Q^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is REAL

A is an array, dimension. (LDA, M) if SIDE = 'L', (LDA, N) if SIDE = 'R' The i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by SGELQF in the first k rows of its array argument A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, K).

TAU Input parameter.

TAU is REAL

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by SGELQF.

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, N); if SIDE = 'R', LWORK >= max(1, M). For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dormlq*. It also exists with a native C interface as *LAPACKE_sormlq*.

4.3.123 sormql

`sormql` overwrites the general real M-by-N matrix C with

```
SIDE = 'L'      SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'T': Q^T * C C * Q^T

where Q is a real orthogonal matrix defined as the product of k elementary reflectors

```
Q = H(k) . . . H(2) H(1)
```

as returned by SGEQLF. Q is of order M if SIDE = 'L' and of order N if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine sormql(SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void sormql_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k, const float *a,
             const armpl_int_t *lda, const float *tau, float *c,
             const armpl_int_t *ldc, float *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'T': Transpose, apply Q^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, K). The i-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by SGEQLF in the last k columns of its array argument A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDA >= max(1, M); if SIDE = 'R', LDA >= max(1, N).

TAU Input parameter.

TAU is REAL

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by SGEQLF.

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, N); if SIDE = 'R', LWORK >= max(1, M). For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dormql*. It also exists with a native C interface as *LAPACKE_sormql*.

4.3.124 sormqr

sormqr overwrites the general real M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'T': Q^T * C C * Q^T

where Q is a real orthogonal matrix defined as the product of k elementary reflectors

```
Q = H(1) H(2) . . . H(k)
```

as returned by SGEQRF. Q is of order M if SIDE = 'L' and of order N if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine sormqr(SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void sormqr_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k, const float *a,
             const armpl_int_t *lda, const float *tau, float *c,
             const armpl_int_t *ldc, float *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'T': Transpose, apply Q^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, K). The i-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,. . . ,k, as returned by SGEQRF in the first k columns of its array argument A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDA >= max(1, M); if SIDE = 'R', LDA >= max(1, N).

TAU Input parameter.

TAU is REAL

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by SGEQRF.

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, N); if SIDE = 'R', LWORK >= max(1, M). For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dormqr*. It also exists with a native C interface as *LAPACKE_sormqr*.

4.3.125 sormrq

sormrq overwrites the general real M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'T': Q^T * C C * Q^T

where Q is a real orthogonal matrix defined as the product of k elementary reflectors

```
Q = H(1) H(2) . . . H(k)
```

as returned by SGERQF. Q is of order M if SIDE = 'L' and of order N if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine sormrq(SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void sormrq_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k, const float *a,
             const armpl_int_t *lda, const float *tau, float *c,
             const armpl_int_t *ldc, float *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'T': Transpose, apply Q^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is REAL

A is an array, dimension. (LDA, M) if SIDE = 'L', (LDA, N) if SIDE = 'R' The i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by SGERQF in the last k rows of its array argument A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, K).

TAU Input parameter.

TAU is REAL

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by SGERQF.

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, N); if SIDE = 'R', LWORK >= max(1, M). For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dormrq*. It also exists with a native C interface as *LAPACKE_sormrq*.

4.3.126 sormrz

sormrz overwrites the general real M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'T': Q^T * C C * Q^T

where Q is a real orthogonal matrix defined as the product of k elementary reflectors

```
Q = H(1) H(2) . . . H(k)
```

as returned by STZRZF. Q is of order M if SIDE = 'L' and of order N if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine sormrz(SIDE, TRANS, M, N, K, L, A, LDA, TAU, C, LDC, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void sormrz_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k, const armpl_int_t *l,
             const float *a, const armpl_int_t *lda, const float *tau,
             float *c, const armpl_int_t *ldc, float *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'T': Transpose, apply Q^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

L Input parameter.

L is INTEGER

The number of columns of the matrix A containing the meaningful part of the Householder reflectors. If SIDE = 'L', M >= L >= 0, if SIDE = 'R', N >= L >= 0.

A Input parameter.

A is REAL

A is an array, dimension. (LDA, M) if SIDE = 'L', (LDA, N) if SIDE = 'R' The i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by STZRZF in the last k rows of its array argument A. A is modified by the routine but restored on exit.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, K).

TAU Input parameter.

TAU is REAL

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by STZRZF.

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^H * C or C*Q^H or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, N); if SIDE = 'R', LWORK >= max(1, M). For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dormrz*. It also exists with a native C interface as *LAPACKE_sormrz*.

4.3.127 spbtrf

spbtrf computes the Cholesky factorization of a real symmetric positive definite band matrix A.

The factorization has the form

```
 A = U**T * U,  if UPLO = 'U', or
A = L  * L**T,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is lower triangular.

Syntax

Fortran specification:

```
use armpl_library

subroutine spbtrf(UPLO, N, KD, AB, LDAB, INFO)
```

C specification:

```
#include "armpl.h"

void spbtrf_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
             float *ab, const armpl_int_t *ldab, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is REAL

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, if INFO = 0, the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T of the band matrix A, in the same storage format as A.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i is not positive definite, and the factorization could not be completed.

Related Information

For this routine in other precisions, please see *cpbtrf*, *dpbtrf* and *zpbtrf*. It also exists with a native C interface as *LAPACKE_spbtrf*.

4.3.128 spftrf

spftrf computes the Cholesky factorization of a real symmetric positive definite matrix A.

The factorization has the form

```
A = U**T * U,  if UPLO = 'U', or
A = L  * L**T,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is lower triangular.

This is the block version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine spftrf(TRANSR, UPLO, N, A, INFO)
```

C specification:

```
#include "armpl.h"

void spftrf_(const char *transr, const char *uplo, const armpl_int_t *n,
             float *a, armpl_int_t *info, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': The Normal TRANSR of RFP A is stored; = 'T': The Transpose TRANSR of RFP A is stored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of RFP A is stored; = 'L': Lower triangle of RFP A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (N*(N+1)/2);. On entry, the symmetric matrix A in RFP format. RFP format is described by TRANSR, UPLO, and N as follows: If TRANSR = 'N' then RFP A is (0:N,0:k-1) when N is even; k=N/2. RFP A is (0:N-1,0:k) when N is odd; k=N/2. IF TRANSR = 'T' then RFP is the transpose of RFP A as defined when TRANSR = 'N'. The contents of RFP A are defined by UPLO as follows: If UPLO = 'U' the RFP A contains the NT elements of upper packed A. If UPLO = 'L' the RFP A contains the elements of lower packed A. The LDA of RFP A is (N+1)/2 when TRANSR = 'T'. When TRANSR is 'N' the LDA is N+1 when N is even and N is odd. See the Note below for more details.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization RFP A = U^T *U or RFP A = L*L^T .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i is not positive definite, and the factorization could not be completed.

Related Information

For this routine in other precisions, please see *cpftrf*, *dpftrf* and *zpftrf*. It also exists with a native C interface as *LAPACKE_spftrf*.

4.3.129 spotrf

spotrf computes the Cholesky factorization of a real symmetric positive definite matrix A.

The factorization has the form

```
A = U**T * U,  if UPLO = 'U', or
A = L  * L**T,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is lower triangular.

This is the block version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine spotrf(UPLO, N, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void spotrf_(const char *uplo, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T .

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i is not positive definite, and the factorization could not be completed.

Related Information

For this routine in other precisions, please see *cpotrf*, *dpotrf* and *zpotrf*. It also exists with a native C interface as *LAPACKE_spotrf*.

4.3.130 spotrf2

spotrf2 computes the Cholesky factorization of a real symmetric positive definite matrix A using the recursive algorithm.

The factorization has the form

```
A = U**T * U,  if UPLO = 'U', or
A = L  * L**T,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is lower triangular.

This is the recursive version of the algorithm. It divides the matrix into four submatrices:

```
    [  A11 | A12  ]  where A11 is n1 by n1 and A22 is n2 by n2
A = [ -----|----- ]  with n1 = n/2
    [  A21 | A22  ]       n2 = n-n1
```

The subroutine calls itself to factor A11. Update and scale A21 or A12, update A22 then call itself to factor A22.

Syntax

Fortran specification:

```
use armpl_library

recursive subroutine spotrf2(UPLO, N, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void spotrf2_(const char *uplo, const armpl_int_t *n, float *a,
              const armpl_int_t *lda, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T .

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i is not positive definite, and the factorization could not be completed.

Related Information

For this routine in other precisions, please see *cpotrf2*, *dpotrf2* and *zpotrf2*. It also exists with a native C interface as *LAPACKE_spotrf2*.

4.3.131 spptrf

spptrf computes the Cholesky factorization of a real symmetric positive definite matrix A stored in packed format.

The factorization has the form

```
A = U**T * U,  if UPLO = 'U', or
A = L  * L**T,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is lower triangular.

Syntax

Fortran specification:

```
use armpl_library

subroutine spptrf(UPLO, N, AP, INFO)
```

C specification:

```
#include "armpl.h"

void spptrf_(const char *uplo, const armpl_int_t *n, float *ap,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is REAL

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n. See below for further details.

On exit, if INFO = 0, the triangular factor U or L from the Cholesky factorization $A = U^T * U$ or $A = L * L^T$, in the same storage format as A.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i is not positive definite, and the factorization could not be completed.

Related Information

For this routine in other precisions, please see *cpptrf*, *dpptrf* and *zpptrf*. It also exists with a native C interface as *LAPACKE_spptrf*.

4.3.132 spstrf

spstrf computes the Cholesky factorization with complete pivoting of a real symmetric positive semidefinite matrix A.

The factorization has the form

```
P**T * A * P = U**T * U ,  if UPLO = 'U',
P**T * A * P = L  * L**T,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is lower triangular, and P is stored as vector PIV.

This algorithm does not attempt to check that A is positive semidefinite. This version of the algorithm calls level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine spstrf(UPLO, N, A, LDA, PIV, RANK, TOL, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void spstrf_(const char *uplo, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, armpl_int_t *piv, armpl_int_t *rank,
             const float *tol, float *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored. = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading n by n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n by n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization as above.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

PIV Output parameter.

PIV is INTEGER array, dimension (N)

PIV is such that the nonzero entries are P(PIV(K), K) = 1.

RANK Output parameter.

RANK is INTEGER

The rank of A given by the number of steps the algorithm completed.

TOL Input parameter.

TOL is REAL

User defined tolerance. If TOL < 0, then N*U*MAX(A(K,K)) will be used. The algorithm terminates at the (K-1)st step if the pivot <= TOL.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (2*N). Work space.

INFO Output parameter.

INFO is INTEGER

< 0: If INFO = -K, the K-th argument had an illegal value, = 0: algorithm completed successfully, and > 0: the matrix A is either rank deficient with computed rank as returned in RANK, or is not positive semidefinite. See Section 7 of LAPACK Working Note #161 for further information.

Related Information

For this routine in other precisions, please see *cpstrf*, *dpstrf* and *zpstrf*. It also exists with a native C interface as *LAPACKE_spstrf*.

4.3.133 spttrf

`spttrf` computes the L*D*L^T factorization of a real symmetric positive definite tridiagonal matrix A. The factorization may also be regarded as having the form A = U^T *D*U.

Syntax

Fortran specification:

```
use armpl_library

subroutine spttrf(N, D, E, INFO)
```

C specification:

```
#include "armpl.h"

void spttrf_(const armpl_int_t *n, float *d, float *e, armpl_int_t *info);
```

Parameters

N Input parameter.

`N` is `INTEGER`

The order of the matrix A. `N` >= 0.

D Input and output parameter.

`D` is `REAL`

`D` is an array, dimension (`N`). On entry, the n diagonal elements of the tridiagonal matrix A. On exit, the n diagonal elements of the diagonal matrix `D` from the L*D*L^T factorization of A.

E Input and output parameter.

`E` is `REAL`

`E` is an array, dimension (N-1). On entry, the (n-1) subdiagonal elements of the tridiagonal matrix A. On exit, the (n-1) subdiagonal elements of the unit bidiagonal factor L from the L*D*L^T factorization of A. `E` can also be regarded as the superdiagonal of the unit bidiagonal factor U from the U^T *D*U factorization of A.

INFO Output parameter.

`INFO` is `INTEGER`

= 0: successful exit < 0: if `INFO` = -k, the k-th argument had an illegal value > 0: if `INFO` = k, the leading minor of order k is not positive definite; if k < `N`, the factorization could not be completed, while if k = `N`, the factorization was completed, but D(`N`) <= 0.

Related Information

For this routine in other precisions, please see *cpttrf*, *dpttrf* and *zpttrf*. It also exists with a native C interface as *LAPACKE_spttrf*.

4.3.134 ssptrf

ssptrf computes the factorization of a real symmetric matrix A stored in packed format using the Bunch-Kaufman diagonal pivoting method:

```
A = U*D*U**T  or  A = L*D*L**T
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssptrf(UPLO, N, AP, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void ssptrf_(const char *uplo, const armpl_int_t *n, float *ap,
             armpl_int_t *ipiv, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is REAL

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L, stored as a packed triangular matrix overwriting A (see below for further details).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *csptrf*, *dsptrf* and *zsptrf*. It also exists with a native C interface as *LAPACKE_ssptrf*.

4.3.135 ssytrf

ssytrf computes the factorization of a real symmetric matrix A using the Bunch-Kaufman diagonal pivoting method. The form of the factorization is

```
A = U*D*U**T  or  A = L*D*L**T
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssytrf(UPLO, N, A, LDA, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ssytrf_(const char *uplo, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, armpl_int_t *ipiv, float *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L (see below for further details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >=1. For best performance LWORK >= N*NB, where NB is the block size returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *csytrf*, *dsytrf* and *zsytrf*. It also exists with a native C interface as *LAPACKE_ssytrf*.

4.3.136 ssytrf_aa

SSYTRF_AA computes the factorization of a real symmetric matrix A using the Aasen's algorithm. The form of the factorization is

```
A = U*T*U**T  or  A = L*T*L**T
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and T is a symmetric tridiagonal matrix.

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssytrf_aa(UPLO, N, A, LDA, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ssytrf_aa_(const char *uplo, const armpl_int_t *n, float *a,
                const armpl_int_t *lda, armpl_int_t *ipiv, float *work,
                const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, the tridiagonal matrix is stored in the diagonals and the subdiagonals of A just below (or above) the diagonals, and L is stored below (or above) the subdiaonals, when UPLO is 'L' (or 'U').

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

On exit, it contains the details of the interchanges, i.e., the row and column k of A were interchanged with the row and column IPIV(k).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= MAX(1,2*N). For optimum performance LWORK >= N*(1+NB), where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *csytrf_aa*, *dsytrf_aa* and *zsytrf_aa*. It also exists with a native C interface as *LAPACKE_ssytrf_aa*.

4.3.137 ssytrf_rk

SSYTRF_RK computes the factorization of a real symmetric matrix A using the bounded Bunch-Kaufman (rook) diagonal pivoting method:

```
A = P*U*D*(U**T)*(P**T) or A = P*L*D*(L**T)*(P**T),
```

where U (or L) is unit upper (or lower) triangular matrix, U^T (or L^T) is the transpose of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the blocked version of the algorithm, calling Level 3 BLAS. For more information see Further Details section.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssytrf_rk(UPLO, N, A, LDA, E, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ssytrf_rk_(const char *uplo, const armpl_int_t *n, float *a,
                const armpl_int_t *lda, float *e, armpl_int_t *ipiv,
                float *work, const armpl_int_t *lwork, armpl_int_t *info,
                ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U': the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced.

If UPLO = 'L': the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, contains: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Output parameter.

E is REAL

E is an array, dimension (N). On exit, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) is set to 0; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) is set to 0.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is set to 0 in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

IPIV describes the permutation matrix P in the factorization of matrix A as follows. The absolute value of IPIV(k) represents the index of row and column that were interchanged with the k-th row and column. The value of UPLO describes the order in which the interchanges were applied. Also, the sign of IPIV represents the block structure of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks which correspond to 1 or 2 interchanges at each factorization step. For more info see Further Details section.

If UPLO = 'U', (in factorization order, k decreases from N to 1): a) A single positive entry IPIV(k) > 0 means: D(k,k) is a 1-by-1 diagonal block. If IPIV(k) != k, rows and columns k and IPIV(k) were interchanged in the matrix A(1:N,1:N); If IPIV(k) = k, no interchange occurred.

b) A pair of consecutive negative entries IPIV(k) < 0 and IPIV(k-1) < 0 means: D(k-1:k,k-1:k) is a 2-by-2 diagonal block. (NOTE: negative entries in IPIV appear ONLY in pairs). 1) If -IPIV(k) != k, rows and columns k and -IPIV(k) were interchanged in the matrix A(1:N,1:N). If -IPIV(k) = k, no interchange occurred. 2) If -IPIV(k-1) != k-1, rows and columns k-1 and -IPIV(k-1) were interchanged in the matrix A(1:N,1:N). If -IPIV(k-1) = k-1, no interchange occurred.

c) In both cases a) and b), always ABS(IPIV(k)) <= k.

d) NOTE: Any entry IPIV(k) is always NONZERO on output.

If UPLO = 'L', (in factorization order, k increases from 1 to N): a) A single positive entry IPIV(k) > 0 means: D(k,k) is a 1-by-1 diagonal block. If IPIV(k) != k, rows and columns k and IPIV(k) were interchanged in the matrix A(1:N,1:N). If IPIV(k) = k, no interchange occurred.

b) A pair of consecutive negative entries IPIV(k) < 0 and IPIV(k+1) < 0 means: D(k:k+1,k:k+1) is a 2-by-2 diagonal block. (NOTE: negative entries in IPIV appear ONLY in pairs). 1) If -IPIV(k) != k, rows and columns k and -IPIV(k) were interchanged in the matrix A(1:N,1:N). If -IPIV(k) = k, no interchange occurred. 2) If -IPIV(k+1) != k+1, rows and columns k-1 and -IPIV(k-1) were interchanged in the matrix A(1:N,1:N). If -IPIV(k+1) = k+1, no interchange occurred.

c) In both cases a) and b), always ABS(IPIV(k)) >= k.

d) NOTE: Any entry IPIV(k) is always NONZERO on output.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)).. On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >=1. For best performance LWORK >= N*NB, where NB is the block size returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit

< 0: If INFO = -k, the k-th argument had an illegal value

> 0: If INFO = k, the matrix A is singular, because: If UPLO = 'U': column k in the upper triangular part of A contains all zeros. If UPLO = 'L': column k in the lower triangular part of A contains all zeros.

Therefore D(k,k) is exactly zero, and superdiagonal elements of column k of U (or subdiagonal elements of column k of L) are all zeros. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

NOTE: INFO only stores the first occurrence of a singularity, any subsequent occurrence of singularity is not stored in INFO even though the factorization always completes.

Related Information

For this routine in other precisions, please see *csytrf_rk*, *dsytrf_rk* and *zsytrf_rk*. It also exists with a native C interface as *LAPACKE_ssytrf_rk*.

4.3.138 ssytrf_rook

SSYTRF_ROOK computes the factorization of a real symmetric matrix A using the bounded Bunch-Kaufman ("rook") diagonal pivoting method. The form of the factorization is

```
A = U*D*U**T  or  A = L*D*L**T
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssytrf_rook(UPLO, N, A, LDA, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ssytrf_rook_(const char *uplo, const armpl_int_t *n, float *a,
                  const armpl_int_t *lda, armpl_int_t *ipiv, float *work,
                  const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L (see below for further details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D.

If UPLO = 'U': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k-1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k-1 and -IPIV(k-1) were inerchaged, D(k-1:k,k-1:k) is a 2-by-2 diagonal block.

If UPLO = 'L': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k+1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k+1 and -IPIV(k+1) were inerchaged, D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)).. On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >=1. For best performance LWORK >= N*NB, where NB is the block size returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *csytrf_rook*, *dsytrf_rook* and *zsytrf_rook*. It also exists with a native C interface as *LAPACKE_ssytrf_rook*.

4.3.139 stplqt

DTPLQT computes a blocked LQ factorization of a real "triangular-pentagonal" matrix C, which is composed of a triangular block A and pentagonal block B, using the compact WY representation for Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine stplqt(M, N, L, MB, A, LDA, B, LDB, T, LDT, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void stplqt_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *l,
             const armpl_int_t *mb, float *a, const armpl_int_t *lda,
             float *b, const armpl_int_t *ldb, float *t,
             const armpl_int_t *ldt, float *work, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix B, and the order of the triangular matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix B. N >= 0.

L Input parameter.

L is INTEGER

The number of rows of the lower trapezoidal part of B. MIN(M, N) >= L >= 0. See Further Details.

MB Input parameter.

MB is INTEGER

The block size to be used in the blocked QR. M >= MB >= 1.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, M). On entry, the lower triangular M-by-M matrix A. On exit, the elements on and below the diagonal of the array contain the lower triangular matrix L.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, N). On entry, the pentagonal M-by-N matrix B. The first N-L columns are rectangular, and the last L columns are lower trapezoidal. On exit, B contains the pentagonal matrix V. See Further Details.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M).

T Output parameter.

T is REAL

T is an array, dimension (LDT, N). The lower triangular block reflectors stored in compact form as a sequence of upper triangular blocks. See Further Details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= MB.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MB*M) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctplqt*, *dtplqt* and *ztplqt*.

4.3.140 stpmlqt

DTPMQRT applies a real orthogonal matrix Q obtained from a "triangular-pentagonal" real block reflector H to a general real matrix C, which consists of two blocks A and B.

Syntax

Fortran specification:

```
use armpl_library

subroutine stpmlqt(SIDE, TRANS, M, N, K, L, MB, V, LDV, T, LDT, A, LDA, B,
                   LDB, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void stpmlqt_(const char *side, const char *trans, const armpl_int_t *m,
              const armpl_int_t *n, const armpl_int_t *k,
              const armpl_int_t *l, const armpl_int_t *mb, const float *v,
              const armpl_int_t *ldv, const float *t, const armpl_int_t *ldt,
              float *a, const armpl_int_t *lda, float *b,
              const armpl_int_t *ldb, float *work, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'T': Transpose, apply Q^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix B. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix B. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q.

L Input parameter.

L is INTEGER

The order of the trapezoidal part of V. K >= L >= 0. See Further Details.

MB Input parameter.

MB is INTEGER

The block size used for the storage of T. K >= MB >= 1. This must be the same value of MB used to generate T in DTPLQT.

V Input parameter.

V is REAL

V is an array, dimension (LDA, K). The i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,. . . ,k, as returned by DTPLQT in B. See Further Details.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. If SIDE = 'L', LDV >= max(1, M); if SIDE = 'R', LDV >= max(1, N).

T Input parameter.

T is REAL

T is an array, dimension (LDT, K). The upper triangular factors of the block reflectors as returned by DTPLQT, stored as a MB-by-K matrix.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= MB.

A Input and output parameter.

A is REAL

A is an array, dimension. (LDA, N) if SIDE = 'L' or (LDA, K) if SIDE = 'R' On entry, the K-by-N or M-by-K matrix A. On exit, A is overwritten by the corresponding block of Q*C or Q^T *C or C*Q or C*Q^T . See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDC >= max(1, K); If SIDE = 'R', LDC >= max(1, M).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, N). On entry, the M-by-N matrix B. On exit, B is overwritten by the corresponding block of Q*C or Q^T *C or C*Q or C*Q^T . See Further Details.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M).

WORK Output parameter.

WORK is REAL array. The dimension of WORK is

N* MB if SIDE = 'L', or M* MB if SIDE = 'R'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctpmlqt*, *dtpmlqt* and *ztpmlqt*.

4.3.141 stpmqrt

stpmqrt applies a real orthogonal matrix Q obtained from a "triangular-pentagonal" real block reflector H to a general real matrix C, which consists of two blocks A and B.

Syntax

Fortran specification:

```
use armpl_library

subroutine stpmqrt(SIDE, TRANS, M, N, K, L, NB, V, LDV, T, LDT, A, LDA, B,
                   LDB, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void stpmqrt_(const char *side, const char *trans, const armpl_int_t *m,
              const armpl_int_t *n, const armpl_int_t *k,
              const armpl_int_t *l, const armpl_int_t *nb, const float *v,
              const armpl_int_t *ldv, const float *t, const armpl_int_t *ldt,
              float *a, const armpl_int_t *lda, float *b,
              const armpl_int_t *ldb, float *work, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'T': Transpose, apply Q^T.

M Input parameter.

M is INTEGER

The number of rows of the matrix B. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix B. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q.

L Input parameter.

L is INTEGER

The order of the trapezoidal part of V. K >= L >= 0. See Further Details.

NB Input parameter.

NB is INTEGER

The block size used for the storage of T. K >= NB >= 1. This must be the same value of NB used to generate T in CTPQRT.

V Input parameter.

V is REAL

V is an array, dimension (LDA, K). The i-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,. . . ,k, as returned by CTPQRT in B. See Further Details.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. If SIDE = 'L', LDV >= max(1, M); if SIDE = 'R', LDV >= max(1, N).

T Input parameter.

T is REAL

T is an array, dimension (LDT, K). The upper triangular factors of the block reflectors as returned by CTPQRT, stored as a NB-by-K matrix.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= NB.

A Input and output parameter.

A is REAL

A is an array, dimension. (LDA, N) if SIDE = 'L' or (LDA, K) if SIDE = 'R' On entry, the K-by-N or M-by-K matrix A. On exit, A is overwritten by the corresponding block of Q*C or Q^T*C or C*Q or C*Q^T. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDC >= max(1, K); If SIDE = 'R', LDC >= max(1, M).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, N). On entry, the M-by-N matrix B. On exit, B is overwritten by the corresponding block of Q*C or Q^T*C or C*Q or C*Q^T. See Further Details.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M).

WORK Output parameter.

WORK is REAL array. The dimension of WORK is

N* NB if SIDE = 'L', or M* NB if SIDE = 'R'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctpmqrt*, *dtpmqrt* and *ztpmqrt*. It also exists with a native C interface as *LAPACKE_stpmqrt*.

4.3.142 stpqrt

stpqrt computes a blocked QR factorization of a real "triangular-pentagonal" matrix C, which is composed of a triangular block A and pentagonal block B, using the compact WY representation for Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine stpqrt(M, N, L, NB, A, LDA, B, LDB, T, LDT, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void stpqrt_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *l,
             const armpl_int_t *nb, float *a, const armpl_int_t *lda,
             float *b, const armpl_int_t *ldb, float *t,
             const armpl_int_t *ldt, float *work, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix B. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix B, and the order of the triangular matrix A. N >= 0.

L Input parameter.

L is INTEGER

The number of rows of the upper trapezoidal part of B. MIN(M, N) >= L >= 0. See Further Details.

NB Input parameter.

NB is INTEGER

The block size to be used in the blocked QR. N >= NB >= 1.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the upper triangular N-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the upper triangular matrix R.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, N). On entry, the pentagonal M-by-N matrix B. The first M-L rows are rectangular, and the last L rows are upper trapezoidal. On exit, B contains the pentagonal matrix V. See Further Details.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M).

T Output parameter.

T is REAL

T is an array, dimension (LDT, N). The upper triangular block reflectors stored in compact form as a sequence of upper triangular blocks. See Further Details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= NB.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (NB*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctpqrt*, *dtpqrt* and *ztpqrt*. It also exists with a native C interface as *LAPACKE_stpqrt*.

4.3.143 stzrzf

stzrzf reduces the M-by-N (M<=N) real upper trapezoidal matrix A to upper triangular form by means of orthogonal transformations.

The upper trapezoidal matrix A is factored as

```
A = ( R  0 ) * Z,
```

where Z is an N-by-N orthogonal matrix and R is an M-by-M upper triangular matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine stzrzf(M, N, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void stzrzf_(const armpl_int_t *m, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, float *tau, float *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= M.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the leading M-by-N upper trapezoidal part of the array A must contain the matrix to be factorized. On exit, the leading M-by-M upper triangular part of A contains the upper triangular matrix R, and elements M+1 to N of the first M rows of A, with the array TAU, represent the orthogonal matrix Z as a product of M elementary reflectors.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is REAL

TAU is an array, dimension (M). The scalar factors of the elementary reflectors.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, M). For optimum performance LWORK >= M*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctzrzf*, *dtzrzf* and *ztzrzf*. It also exists with a native C interface as *LAPACKE_stzrzf*.

4.3.144 zgbtrf

zgbtrf computes an LU factorization of a complex m-by-n band matrix A using partial pivoting with row interchanges.

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgbtrf(M, N, KL, KU, AB, LDAB, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void zgbtrf_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *kl, const armpl_int_t *ku,
             armpl_doublecomplex_t *ab, const armpl_int_t *ldab,
             armpl_int_t *ipiv, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

AB Input and output parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows KL+1 to 2*KL+KU+1; rows 1 to KL of the array need not be set. The j-th column of A is stored in the j-th column of the array AB as follows: AB(kl+ku+1+i-j,j) = A(i,j) for max(1,j-ku)<=i<=min(m,j+kl)

On exit, details of the factorization: U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1. See below for further details.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= 2*KL+KU+1.

IPIV Output parameter.

IPIV is INTEGER array, dimension (min(M, N))

The pivot indices; for 1 <= i <= min(M, N), row i of the matrix was interchanged with row IPIV(i).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = +i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *cgbtrf*, *dgbtrf* and *sgbtrf*. It also exists with a native C interface as *LAPACKE_zgbtrf*.

4.3.145 zgelq

zgelq computes a LQ factorization of an M-by-N matrix A.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgelq(M, N, A, LDA, T, TSIZE, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgelq_(const armpl_int_t *m, const armpl_int_t *n,
            armpl_doublecomplex_t *a, const armpl_int_t *lda,
            armpl_doublecomplex_t *t, const armpl_int_t *tsize,
            armpl_doublecomplex_t *work, const armpl_int_t *lwork,
            armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and below the diagonal of the array contain the M-by-min(M, N) lower trapezoidal matrix L (L is lower triangular if M <= N); the elements above the diagonal are used to store part of the data structure to represent Q.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

T Output parameter.

T is COMPLEX*16

T is an array, dimension (MAX(5, TSIZE)). On exit, if INFO = 0, T(1) returns optimal (or either minimal or optimal, if query is assumed) TSIZE. See TSIZE for details. Remaining T contains part of the data structure used to represent Q. If one wants to apply or construct Q, then one needs to keep T (in addition to A) and pass it to further subroutines.

TSIZE Input parameter.

TSIZE is INTEGER

If TSIZE >= 5, the dimension of the array T. If TSIZE = -1 or -2, then a workspace query is assumed. The routine only calculates the sizes of the T and WORK arrays, returns these values as the first entries of the T and WORK arrays, and no error message related to T or WORK is issued by XERBLA. If TSIZE = -1, the routine calculates optimal size of T for the optimum performance and returns this value in T(1). If TSIZE = -2, the routine calculates minimal size of T and returns this value in T(1).

WORK Output parameter.

(workspace) COMPLEX*16

(workspace) is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) contains optimal (or either minimal or optimal, if query was assumed) LWORK. See LWORK for details.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If LWORK = -1 or -2, then a workspace query is assumed. The routine only calculates the sizes of the T and WORK arrays, returns these values as the first entries of the T and WORK arrays, and no error message related to T or WORK is issued by XERBLA. If LWORK = -1, the routine calculates optimal size of WORK for the optimal performance and returns this value in WORK(1). If LWORK = -2, the routine calculates minimal size of WORK and returns this value in WORK(1).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgelq*, *dgelq* and *sgelq*. It also exists with a native C interface as *LAPACKE_zgelq*.

4.3.146 zgelqf

zgelqf computes an LQ factorization of a complex M-by-N matrix A: A = L * Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgelqf(M, N, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgelqf_(const armpl_int_t *m, const armpl_int_t *n,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_doublecomplex_t *tau, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and below the diagonal of the array contain the m-by-min(m,n) lower trapezoidal matrix L (L is lower triangular if m <= n); the elements above the diagonal, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is COMPLEX*16

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, M). For optimum performance LWORK >= M*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgelqf*, *dgelqf* and *sgelqf*. It also exists with a native C interface as *LAPACKE_zgelqf*.

4.3.147 zgelqt

`zgelqt` computes a blocked LQ factorization of a complex M-by-N matrix A using the compact WY representation of Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgelqt(M, N, MB, A, LDA, T, LDT, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgelqt_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *mb, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_doublecomplex_t *t,
             const armpl_int_t *ldt, armpl_doublecomplex_t *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

`M` is `INTEGER`

The number of rows of the matrix `A`. `M` >= 0.

N Input parameter.

`N` is `INTEGER`

The number of columns of the matrix `A`. `N` >= 0.

MB Input parameter.

`MB` is `INTEGER`

The block size to be used in the blocked QR. MIN(`M`, `N`) >= `MB` >= 1.

A Input and output parameter.

`A` is `COMPLEX*16`

`A` is an array, dimension (`LDA`, `N`). On entry, the M-by-N matrix `A`. On exit, the elements on and below the diagonal of the array contain the M-by-MIN(`M`, `N`) lower trapezoidal matrix L (L is lower triangular if `M` <= N); the elements above the diagonal are the rows of V.

LDA Input parameter.

`LDA` is `INTEGER`

The leading dimension of the array `A`. `LDA` >= max(1, `M`).

T Output parameter.

T is COMPLEX*16

T is an array, dimension (LDT,MIN(M, N)). The upper triangular block reflectors stored in compact form as a sequence of upper triangular blocks. See below for further details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= MB.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MB*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgelqt*, *dgelqt* and *sgelqt*.

4.3.148 zgemlq

zgemlq overwrites the general real M-by-N matrix C with SIDE = 'L' SIDE = 'R' TRANS = 'N': Q * C C * Q TRANS = 'C': Q^H * C C * Q^H where Q is a complex unitary matrix defined as the product of blocked elementary reflectors computed by short wide LQ factorization (ZGELQ)

Syntax

Fortran specification:

```
use armpl_library

subroutine zgemlq(SIDE, TRANS, M, N, K, A, LDA, T, TSIZE, C, LDC, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void zgemlq_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_doublecomplex_t *t, const armpl_int_t *tsize,
             armpl_doublecomplex_t *c, const armpl_int_t *ldc,
             armpl_doublecomplex_t *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'T': Transpose, apply Q^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >=0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension. (LDA, M) if SIDE = 'L', (LDA, N) if SIDE = 'R' Part of the data structure to represent Q as returned by ZGELQ.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, K).

T Input parameter.

T is COMPLEX*16

T is an array, dimension (MAX(5, TSIZE)).. Part of the data structure to represent Q as returned by ZGELQ.

TSIZE Input parameter.

TSIZE is INTEGER

The dimension of the array T. TSIZE >= 5.

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

(workspace) COMPLEX*16

(workspace) is an array, dimension (MAX(1, LWORK)) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If LWORK = -1, then a workspace query is assumed. The routine only calculates the size of the WORK array, returns this value as WORK(1), and no error message related to WORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgemlq*, *dgemlq* and *sgemlq*. It also exists with a native C interface as *LAPACKE_zgemlq*.

4.3.149 zgemlqt

zgemlqt overwrites the general real M-by-N matrix C with

```
SIDE = 'L'      SIDE = 'R'
```

TRANS = 'N': Q C C Q TRANS = 'C': Q^H C C Q^H

where Q is a complex orthogonal matrix defined as the product of K elementary reflectors:

```
Q = H(1) H(2) . . . H(K) = I - V T V**H
```

generated using the compact WY representation as returned by ZGELQT.

Q is of order M if SIDE = 'L' and of order N if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgemlqt(SIDE, TRANS, M, N, K, MB, V, LDV, T, LDT, C, LDC, WORK,
                   INFO)
```

C specification:

```
#include "armpl.h"

void zgemlqt_(const char *side, const char *trans, const armpl_int_t *m,
              const armpl_int_t *n, const armpl_int_t *k,
              const armpl_int_t *mb, const armpl_doublecomplex_t *v,
              const armpl_int_t *ldv, const armpl_doublecomplex_t *t,
              const armpl_int_t *ldt, armpl_doublecomplex_t *c,
              const armpl_int_t *ldc, armpl_doublecomplex_t *work,
              armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left; = 'R': apply Q or Q^H from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'C': Transpose, apply Q^H .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

MB Input parameter.

MB is INTEGER

The block size used for the storage of T. K >= MB >= 1. This must be the same value of MB used to generate T in DGELQT.

V Input parameter.

V is COMPLEX*16

V is an array, dimension. (LDV, M) if SIDE = 'L', (LDV, N) if SIDE = 'R' The i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by DGELQT in the first K rows of its array argument A.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. LDV >= max(1, K).

T Input parameter.

T is COMPLEX*16

T is an array, dimension (LDT, K). The upper triangular factors of the block reflectors as returned by DGELQT, stored as a MB-by-K matrix.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= MB.

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q C, Q^H C, C Q^H or C Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX*16 array. The dimension of

WORK is N* MB if SIDE = 'L', or M* MB if SIDE = 'R'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgemlqt*, *dgemlqt* and *sgemlqt*.

4.3.150 zgemqr

zgemqr overwrites the general real M-by-N matrix C with SIDE = 'L' SIDE = 'R' TRANS = 'N': Q * C C * Q TRANS = 'T': Q^H * C C * Q^H where Q is a complex unitary matrix defined as the product of blocked elementary reflectors computed by tall skinny QR factorization (ZGEQR)

Syntax

Fortran specification:

```
use armpl_library

subroutine zgemqr(SIDE, TRANS, M, N, K, A, LDA, T, TSIZE, C, LDC, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void zgemqr_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_doublecomplex_t *t, const armpl_int_t *tsize,
             armpl_doublecomplex_t *c, const armpl_int_t *ldc,
             armpl_doublecomplex_t *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'T': Transpose, apply Q^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >=0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, K). Part of the data structure to represent Q as returned by ZGEQR.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDA >= max(1, M); if SIDE = 'R', LDA >= max(1, N).

T Input parameter.

T is COMPLEX*16

T is an array, dimension (MAX(5, TSIZE)).. Part of the data structure to represent Q as returned by ZGEQR.

TSIZE Input parameter.

TSIZE is INTEGER

The dimension of the array T. TSIZE >= 5.

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

(workspace) COMPLEX*16

(workspace) is an array, dimension (MAX(1, LWORK)) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If LWORK = -1, then a workspace query is assumed. The routine only calculates the size of the WORK array, returns this value as WORK(1), and no error message related to WORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgemqr*, *dgemqr* and *sgemqr*. It also exists with a native C interface as *LAPACKE_zgemqr*.

4.3.151 zgemqrt

`zgemqrt` overwrites the general complex M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': Q C C Q TRANS = 'C': Q^H C C Q^H

where Q is a complex orthogonal matrix defined as the product of K elementary reflectors:

```
Q = H(1) H(2) . . . H(K) = I - V T V**H
```

generated using the compact WY representation as returned by ZGEQRT.

Q is of order M if SIDE = 'L' and of order N if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgemqrt(SIDE, TRANS, M, N, K, NB, V, LDV, T, LDT, C, LDC, WORK,
                   INFO)
```

C specification:

```
#include "armpl.h"

void zgemqrt_(const char *side, const char *trans, const armpl_int_t *m,
              const armpl_int_t *n, const armpl_int_t *k,
              const armpl_int_t *nb, const armpl_doublecomplex_t *v,
              const armpl_int_t *ldv, const armpl_doublecomplex_t *t,
              const armpl_int_t *ldt, armpl_doublecomplex_t *c,
              const armpl_int_t *ldc, armpl_doublecomplex_t *work,
              armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left; = 'R': apply Q or Q^H from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'C': Transpose, apply Q^H .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

NB Input parameter.

NB is INTEGER

The block size used for the storage of T. K >= NB >= 1. This must be the same value of NB used to generate T in CGEQRT.

V Input parameter.

V is COMPLEX*16

V is an array, dimension (LDV, K). The i-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by CGEQRT in the first K columns of its array argument A.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. If SIDE = 'L', LDA >= max(1, M); if SIDE = 'R', LDA >= max(1, N).

T Input parameter.

T is COMPLEX*16

T is an array, dimension (LDT, K). The upper triangular factors of the block reflectors as returned by CGEQRT, stored as a NB-by-N matrix.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= NB.

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q C, Q^H C, C Q^H or C Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX*16 array. The dimension of WORK is

N* NB if SIDE = 'L', or M* NB if SIDE = 'R'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgemqrt*, *dgemqrt* and *sgemqrt*. It also exists with a native C interface as *LAPACKE_zgemqrt*.

4.3.152 zgeqlf

`zgeqlf` computes a QL factorization of a complex M-by-N matrix A: A = Q * L.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgeqlf(M, N, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgeqlf_(const armpl_int_t *m, const armpl_int_t *n,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_doublecomplex_t *tau, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

`M` is `INTEGER`

The number of rows of the matrix `A`. `M` >= 0.

N Input parameter.

`N` is `INTEGER`

The number of columns of the matrix `A`. `N` >= 0.

A Input and output parameter.

`A` is `COMPLEX*16`

`A` is an array, dimension (`LDA`, `N`). On entry, the M-by-N matrix `A`. On exit, if m >= n, the lower triangle of the subarray A(m-n+1:m,1:n) contains the N-by-N lower triangular matrix L; if m <= n, the elements on and below the (n-m)-th superdiagonal contain the M-by-N lower trapezoidal matrix L; the remaining elements, with the array `TAU`, represent the unitary matrix Q as a product of elementary reflectors (see Further Details).

LDA Input parameter.

`LDA` is `INTEGER`

The leading dimension of the array `A`. `LDA` >= max(1, `M`).

TAU Output parameter.

`TAU` is `COMPLEX*16`

`TAU` is an array, dimension (min(`M`, `N`)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

`WORK` is `COMPLEX*16`

`WORK` is an array, dimension (MAX(1, `LWORK`)). On exit, if `INFO` = 0, WORK(1) returns the optimal `LWORK`.

LWORK Input parameter.

`LWORK` is `INTEGER`

The dimension of the array WORK. LWORK >= max(1, N). For optimum performance LWORK >= N*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgeqlf*, *dgeqlf* and *sgeqlf*. It also exists with a native C interface as *LAPACKE_zgeqlf*.

4.3.153 zgeqp3

zgeqp3 computes a QR factorization with column pivoting of a matrix A: A*P = Q*R using Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgeqp3(M, N, A, LDA, JPVT, TAU, WORK, LWORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgeqp3_(const armpl_int_t *m, const armpl_int_t *n,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_int_t *jpvt, armpl_doublecomplex_t *tau,
             armpl_doublecomplex_t *work, const armpl_int_t *lwork,
             double *rwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the upper triangle of the array contains the min(M, N)-by-N upper trapezoidal matrix R; the elements below the diagonal, together with the array TAU, represent the unitary matrix Q as a product of min(M, N) elementary reflectors.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

JPVT Input and output parameter.

JPVT is INTEGER array, dimension (N)

On entry, if JPVT(J).ne.0, the J-th column of A is permuted to the front of A*P (a leading column); if JPVT(J)=0, the J-th column of A is a free column. On exit, if JPVT(J)=K, then the J-th column of A*P was the the K-th column of A.

TAU Output parameter.

TAU is COMPLEX*16

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO=0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= N+1. For optimal performance LWORK >= (N+1)*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cgeqp3*, *dgeqp3* and *sgeqp3*. It also exists with a native C interface as *LAPACKE_zgeqp3*.

4.3.154 zgeqr

zgeqr computes a QR factorization of an M-by-N matrix A.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgeqr(M, N, A, LDA, T, TSIZE, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgeqr_(const armpl_int_t *m, const armpl_int_t *n,
            armpl_doublecomplex_t *a, const armpl_int_t *lda,
            armpl_doublecomplex_t *t, const armpl_int_t *tsize,
            armpl_doublecomplex_t *work, const armpl_int_t *lwork,
            armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the min(M, N)-by-N upper trapezoidal matrix R (R is upper triangular if M >= N); the elements below the diagonal are used to store part of the data structure to represent Q.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

T Output parameter.

T is COMPLEX*16

T is an array, dimension (MAX(5, TSIZE)). On exit, if INFO = 0, T(1) returns optimal (or either minimal or optimal, if query is assumed) TSIZE. See TSIZE for details. Remaining T contains part of the data structure used to represent Q. If one wants to apply or construct Q, then one needs to keep T (in addition to A) and pass it to further subroutines.

TSIZE Input parameter.

TSIZE is INTEGER

If TSIZE >= 5, the dimension of the array T. If TSIZE = -1 or -2, then a workspace query is assumed. The routine only calculates the sizes of the T and WORK arrays, returns these values as the first entries of the T and WORK arrays, and no error message related to T or WORK is issued by XERBLA. If TSIZE = -1, the routine calculates optimal size of T for the optimum performance and returns this value in T(1). If TSIZE = -2, the routine calculates minimal size of T and returns this value in T(1).

WORK Output parameter.

(workspace) COMPLEX*16

(workspace) is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) contains optimal (or either minimal or optimal, if query was assumed) LWORK. See LWORK for details.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If LWORK = -1 or -2, then a workspace query is assumed. The routine only calculates the sizes of the T and WORK arrays, returns these values as the first entries of the T and

WORK arrays, and no error message related to T or WORK is issued by XERBLA. If LWORK = -1, the routine calculates optimal size of WORK for the optimal performance and returns this value in WORK(1). If LWORK = -2, the routine calculates minimal size of WORK and returns this value in WORK(1).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgeqr*, *dgeqr* and *sgeqr*. It also exists with a native C interface as *LAPACKE_zgeqr*.

4.3.155 zgeqrf

zgeqrf computes a QR factorization of a complex M-by-N matrix A: A = Q * R.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgeqrf(M, N, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgeqrf_(const armpl_int_t *m, const armpl_int_t *n,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_doublecomplex_t *tau, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the min(M, N)-by-N upper trapezoidal matrix R (R is upper triangular if m >= n); the elements below the diagonal, with the array TAU, represent the unitary matrix Q as a product of min(m,n) elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is COMPLEX*16

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N). For optimum performance LWORK >= N*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgeqrf*, *dgeqrf* and *sgeqrf*. It also exists with a native C interface as *LAPACKE_zgeqrf*.

4.3.156 zgeqrfp

zgeqrfp computes a QR factorization of a complex M-by-N matrix A: A = Q * R. The diagonal entries of R are real and nonnegative.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgeqrfp(M, N, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgeqrfp_(const armpl_int_t *m, const armpl_int_t *n,
              armpl_doublecomplex_t *a, const armpl_int_t *lda,
              armpl_doublecomplex_t *tau, armpl_doublecomplex_t *work,
              const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the min(M, N)-by-N upper trapezoidal matrix R (R is upper triangular if m >= n). The diagonal entries of R are real and nonnegative; The elements below the diagonal, with the array TAU, represent the unitary matrix Q as a product of min(m,n) elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is COMPLEX*16

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N). For optimum performance LWORK >= N*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgeqrfp*, *dgeqrfp* and *sgeqrfp*. It also exists with a native C interface as *LAPACKE_zgeqrfp*.

4.3.157 zgeqrt

zgeqrt computes a blocked QR factorization of a complex M-by-N matrix A using the compact WY representation of Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgeqrt(M, N, NB, A, LDA, T, LDT, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgeqrt_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *nb, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_doublecomplex_t *t,
             const armpl_int_t *ldt, armpl_doublecomplex_t *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

NB Input parameter.

NB is INTEGER

The block size to be used in the blocked QR. MIN(M, N) >= NB >= 1.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the min(M, N)-by-N upper trapezoidal matrix R (R is upper triangular if M >= N); the elements below the diagonal are the columns of V.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

T Output parameter.

T is COMPLEX*16

T is an array, dimension (LDT,MIN(M, N)). The upper triangular block reflectors stored in compact form as a sequence of upper triangular blocks. See below for further details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= NB.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (NB*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgeqrt*, *dgeqrt* and *sgeqrt*. It also exists with a native C interface as *LAPACKE_zgeqrt*.

4.3.158 zgerqf

zgerqf computes an RQ factorization of a complex M-by-N matrix A: A = R * Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgerqf(M, N, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgerqf_(const armpl_int_t *m, const armpl_int_t *n,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_doublecomplex_t *tau, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, if m <= n, the upper triangle of the subarray A(1:m,n-m+1:n) contains the M-by-M upper triangular matrix R; if m >= n, the elements on and above the (m-n)-th subdiagonal contain the M-by-N upper trapezoidal matrix R; the remaining elements, with the array TAU, represent the unitary matrix Q as a product of min(m,n) elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is COMPLEX*16

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, M). For optimum performance LWORK >= M*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgerqf*, *dgerqf* and *sgerqf*. It also exists with a native C interface as *LAPACKE_zgerqf*.

4.3.159 zgetrf

zgetrf computes an LU factorization of a general M-by-N matrix A using partial pivoting with row interchanges.

The factorization has the form

```
A = P * L * U
```

where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if m > n), and U is upper triangular (upper trapezoidal if m < n).

This is the right-looking Level 3 BLAS version of the algorithm.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgetrf(M, N, A, LDA, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void zgetrf_(const armpl_int_t *m, const armpl_int_t *n,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_int_t *ipiv, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the M-by-N matrix to be factored. On exit, the factors L and U from the factorization A = P*L*U; the unit diagonal elements of L are not stored.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

IPIV Output parameter.

IPIV is INTEGER array, dimension (min(M, N))

The pivot indices; for 1 <= i <= min(M, N), row i of the matrix was interchanged with row IPIV(i).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *cgetrf*, *dgetrf* and *sgetrf*. It also exists with a native C interface as *LAPACKE_zgetrf*.

4.3.160 zgetrf2

zgetrf2 computes an LU factorization of a general M-by-N matrix A using partial pivoting with row interchanges.

The factorization has the form

```
A = P * L * U
```

where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if m > n), and U is upper triangular (upper trapezoidal if m < n).

This is the recursive version of the algorithm. It divides the matrix into four submatrices:

```
    [  A11 | A12  ]  where A11 is n1 by n1 and A22 is n2 by n2
A = [ -----|----- ]  with n1 = min(m,n)/2
    [  A21 | A22  ]       n2 = n-n1

                                  [ A11 ]
```

The subroutine calls itself to factor [—],

```
                    [ A12 ]
[ A12 ]
```

do the swaps on [—], solve A12, update A22,

```
[ A22 ]
```

then calls itself to factor A22 and do the swaps on A21.

Syntax

Fortran specification:

```
use armpl_library

recursive subroutine zgetrf2(M, N, A, LDA, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void zgetrf2_(const armpl_int_t *m, const armpl_int_t *n,
              armpl_doublecomplex_t *a, const armpl_int_t *lda,
              armpl_int_t *ipiv, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the M-by-N matrix to be factored. On exit, the factors L and U from the factorization A = P*L*U; the unit diagonal elements of L are not stored.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

IPIV Output parameter.

IPIV is INTEGER array, dimension (min(M, N))

The pivot indices; for 1 <= i <= min(M, N), row i of the matrix was interchanged with row IPIV(i).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *cgetrf2*, *dgetrf2* and *sgetrf2*. It also exists with a native C interface as *LAPACKE_zgetrf2*.

4.3.161 zggqrf

`zggqrf` computes a generalized QR factorization of an N-by-M matrix A and an N-by-P matrix B:

```
A = Q*R,        B = Q*T*Z,
```

where Q is an N-by-N unitary matrix, Z is a P-by-P unitary matrix, and R and T assume one of the forms:

if N >= M, R = (R11) M , or if N < M, R = (R11 R12) N,

```
(  0  ) N-M                          N   M-N
   M
```

where R11 is upper triangular, and

if N <= P, T = (0 T12) N, or if N > P, T = (T11) N-P,

```
P-N  N                             ( T21 ) P
                                      P
```

where T12 or T21 is upper triangular.

In particular, if B is square and nonsingular, the GQR factorization of A and B implicitly gives the QR factorization of inv(B)*A:

```
inv(B)*A = Z**H * (inv(T)*R)
```

where inv(B) denotes the inverse of the matrix B, and Z^H denotes the conjugate transpose of matrix Z.

Syntax

Fortran specification:

```
use armpl_library

subroutine zggqrf(N, M, P, A, LDA, TAUA, B, LDB, TAUB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zggqrf_(const armpl_int_t *n, const armpl_int_t *m, const armpl_int_t *p,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_doublecomplex_t *taua, armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, armpl_doublecomplex_t *taub,
             armpl_doublecomplex_t *work, const armpl_int_t *lwork,
             armpl_int_t *info);
```

Parameters

N Input parameter.

`N` is `INTEGER`

The number of rows of the matrices `A` and `B`. `N` >= 0.

M Input parameter.

M is INTEGER

The number of columns of the matrix A. M >= 0.

P Input parameter.

P is INTEGER

The number of columns of the matrix B. P >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, M). On entry, the N-by-M matrix A. On exit, the elements on and above the diagonal of the array contain the min(N, M)-by-M upper trapezoidal matrix R (R is upper triangular if N >= M); the elements below the diagonal, with the array TAUA, represent the unitary matrix Q as a product of min(N, M) elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TAUA Output parameter.

TAUA is COMPLEX*16

TAUA is an array, dimension (min(N, M)). The scalar factors of the elementary reflectors which represent the unitary matrix Q (see Further Details).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, P). On entry, the N-by-P matrix B. On exit, if N <= P, the upper triangle of the subarray B(1:N,P-N+1:P) contains the N-by-N upper triangular matrix T; if N > P, the elements on and above the (N-P)-th subdiagonal contain the N-by-P upper trapezoidal matrix T; the remaining elements, with the array TAUB, represent the unitary matrix Z as a product of elementary reflectors (see Further Details).

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

TAUB Output parameter.

TAUB is COMPLEX*16

TAUB is an array, dimension (min(N, P)). The scalar factors of the elementary reflectors which represent the unitary matrix Z (see Further Details).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N, M, P). For optimum performance LWORK >= max(N, M, P)*max(NB1,NB2,NB3), where NB1 is the optimal blocksize for the QR factorization of an N-by-M matrix, NB2 is the optimal blocksize for the RQ factorization of an N-by-P matrix, and NB3 is the optimal blocksize for a call of ZUNMQR.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cggqrf*, *dggqrf* and *sggqrf*. It also exists with a native C interface as *LAPACKE_zggqrf*.

4.3.162 zggrqf

zggrqf computes a generalized RQ factorization of an M-by-N matrix A and a P-by-N matrix B:

```
A = R*Q,        B = Z*T*Q,
```

where Q is an N-by-N unitary matrix, Z is a P-by-P unitary matrix, and R and T assume one of the forms:

if M <= N, R = (0 R12) M, or if M > N, R = (R11) M-N,

```
N-M  M                            ( R21 ) N
                                    N
```

where R12 or R21 is upper triangular, and

if P >= N, T = (T11) N , or if P < N, T = (T11 T12) P,

```
(  0  ) P-N                         P    N-P
   N
```

where T11 is upper triangular.

In particular, if B is square and nonsingular, the GRQ factorization of A and B implicitly gives the RQ factorization of A*inv(B):

```
A*inv(B) = (R*inv(T))*Z**H
```

where inv(B) denotes the inverse of the matrix B, and Z^H denotes the conjugate transpose of the matrix Z.

Syntax

Fortran specification:

```
use armpl_library

subroutine zggrqf(M, P, N, A, LDA, TAUA, B, LDB, TAUB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zggrqf_(const armpl_int_t *m, const armpl_int_t *p, const armpl_int_t *n,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_doublecomplex_t *taua, armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, armpl_doublecomplex_t *taub,
```

(continues on next page)

(continued from previous page)

```
            armpl_doublecomplex_t *work, const armpl_int_t *lwork,
            armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

P Input parameter.

P is INTEGER

The number of rows of the matrix B. P >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrices A and B. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, if M <= N, the upper triangle of the subarray A(1:M,N-M+1:N) contains the M-by-M upper triangular matrix R; if M > N, the elements on and above the (M-N)-th subdiagonal contain the M-by-N upper trapezoidal matrix R; the remaining elements, with the array TAUA, represent the unitary matrix Q as a product of elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAUA Output parameter.

TAUA is COMPLEX*16

TAUA is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors which represent the unitary matrix Q (see Further Details).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). On entry, the P-by-N matrix B. On exit, the elements on and above the diagonal of the array contain the min(P, N)-by-N upper trapezoidal matrix T (T is upper triangular if P >= N); the elements below the diagonal, with the array TAUB, represent the unitary matrix Z as a product of elementary reflectors (see Further Details).

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, P).

TAUB Output parameter.

TAUB is COMPLEX*16

TAUB is an array, dimension (min(P, N)). The scalar factors of the elementary reflectors which represent the unitary matrix Z (see Further Details).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N, M, P). For optimum performance LWORK >= max(N, M, P)*max(NB1,NB2,NB3), where NB1 is the optimal blocksize for the RQ factorization of an M-by-N matrix, NB2 is the optimal blocksize for the QR factorization of a P-by-N matrix, and NB3 is the optimal blocksize for a call of ZUNMRQ.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO=-i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cggrqf*, *dggrqf* and *sggrqf*. It also exists with a native C interface as *LAPACKE_zggrqf*.

4.3.163 zgttrf

zgttrf computes an LU factorization of a complex tridiagonal matrix A using elimination with partial pivoting and row interchanges.

The factorization has the form

```
A = L * U
```

where L is a product of permutation and unit lower bidiagonal matrices and U is upper triangular with nonzeros in only the main diagonal and first two superdiagonals.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgttrf(N, DL, D, DU, DU2, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void zgttrf_(const armpl_int_t *n, armpl_doublecomplex_t *dl,
             armpl_doublecomplex_t *d, armpl_doublecomplex_t *du,
             armpl_doublecomplex_t *du2, armpl_int_t *ipiv,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A.

DL Input and output parameter.

DL is COMPLEX*16

DL is an array, dimension (N-1). On entry, DL must contain the (n-1) sub-diagonal elements of A.

On exit, DL is overwritten by the (n-1) multipliers that define the matrix L from the LU factorization of A.

D Input and output parameter.

D is COMPLEX*16

D is an array, dimension (N). On entry, D must contain the diagonal elements of A.

On exit, D is overwritten by the n diagonal elements of the upper triangular matrix U from the LU factorization of A.

DU Input and output parameter.

DU is COMPLEX*16

DU is an array, dimension (N-1). On entry, DU must contain the (n-1) super-diagonal elements of A.

On exit, DU is overwritten by the (n-1) elements of the first super-diagonal of U.

DU2 Output parameter.

DU2 is COMPLEX*16

DU2 is an array, dimension (N-2). On exit, DU2 is overwritten by the (n-2) elements of the second super-diagonal of U.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices; for 1 <= i <= n, row i of the matrix was interchanged with row IPIV(i). IPIV(i) will always be either i or i+1; IPIV(i) = i indicates a row interchange was not required.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value > 0: if INFO = k, U(k,k) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *cgttrf*, *dgttrf* and *sgttrf*. It also exists with a native C interface as *LAPACKE_zgttrf*.

4.3.164 zhetrf

zhetrf computes the factorization of a complex Hermitian matrix A using the Bunch-Kaufman diagonal pivoting method. The form of the factorization is

```
A = U*D*U**H  or  A = L*D*L**H
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is Hermitian and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhetrf(UPLO, N, A, LDA, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhetrf_(const char *uplo, const armpl_int_t *n, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *ipiv,
             armpl_doublecomplex_t *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L (see below for further details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >=1. For best performance LWORK >= N*NB, where NB is the block size returned by ILAENV.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *chetrf*. It also exists with a native C interface as *LAPACKE_zhetrf*.

4.3.165 zhetrf_aa

ZHETRF_AA computes the factorization of a complex hermitian matrix A using the Aasen's algorithm. The form of the factorization is

```
A = U*T*U**H  or  A = L*T*L**H
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and T is a hermitian tridiagonal matrix.

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhetrf_aa(UPLO, N, A, LDA, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhetrf_aa_(const char *uplo, const armpl_int_t *n,
                armpl_doublecomplex_t *a, const armpl_int_t *lda,
                armpl_int_t *ipiv, armpl_doublecomplex_t *work,
                const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, the tridiagonal matrix is stored in the diagonals and the subdiagonals of A just below (or above) the diagonals, and L is stored below (or above) the subdiaonals, when UPLO is 'L' (or 'U').

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

On exit, it contains the details of the interchanges, i.e., the row and column k of A were interchanged with the row and column IPIV(k).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= MAX(1,2*N). For optimum performance LWORK >= N*(1+NB), where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *chetrf_aa*. It also exists with a native C interface as *LAPACKE_zhetrf_aa*.

4.3.166 zhetrf_rk

ZHETRF_RK computes the factorization of a complex Hermitian matrix A using the bounded Bunch-Kaufman (rook) diagonal pivoting method:

```
A = P*U*D*(U**H)*(P**T) or A = P*L*D*(L**H)*(P**T),
```

where U (or L) is unit upper (or lower) triangular matrix, U^H (or L^H) is the conjugate of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is Hermitian and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the blocked version of the algorithm, calling Level 3 BLAS. For more information see Further Details section.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhetrf_rk(UPLO, N, A, LDA, E, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhetrf_rk_(const char *uplo, const armpl_int_t *n,
                armpl_doublecomplex_t *a, const armpl_int_t *lda,
                armpl_doublecomplex_t *e, armpl_int_t *ipiv,
                armpl_doublecomplex_t *work, const armpl_int_t *lwork,
                armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the Hermitian matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U': the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced.

If UPLO = 'L': the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, contains: a) ONLY diagonal elements of the Hermitian block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Output parameter.

E is COMPLEX*16

E is an array, dimension (N). On exit, contains the superdiagonal (or subdiagonal) elements of the Hermitian block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) is set to 0; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) is set to 0.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is set to 0 in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

IPIV describes the permutation matrix P in the factorization of matrix A as follows. The absolute value of IPIV(k) represents the index of row and column that were interchanged with the k-th row and column. The value of UPLO describes the order in which the interchanges were applied. Also, the sign of IPIV represents the block structure of the Hermitian block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks which correspond to 1 or 2 interchanges at each factorization step. For more info see Further Details section.

If UPLO = 'U', (in factorization order, k decreases from N to 1): a) A single positive entry IPIV(k) > 0 means: D(k,k) is a 1-by-1 diagonal block. If IPIV(k) != k, rows and columns k and IPIV(k) were interchanged in the matrix A(1:N,1:N); If IPIV(k) = k, no interchange occurred.

b) A pair of consecutive negative entries IPIV(k) < 0 and IPIV(k-1) < 0 means: D(k-1:k,k-1:k) is a 2-by-2 diagonal block. (NOTE: negative entries in IPIV appear ONLY in pairs). 1) If -IPIV(k) != k, rows and columns k and -IPIV(k) were interchanged in the matrix A(1:N,1:N). If -IPIV(k) = k, no interchange occurred. 2) If -IPIV(k-1) != k-1, rows and columns k-1 and -IPIV(k-1) were interchanged in the matrix A(1:N,1:N). If -IPIV(k-1) = k-1, no interchange occurred.

c) In both cases a) and b), always ABS(IPIV(k)) <= k.

d) NOTE: Any entry IPIV(k) is always NONZERO on output.

If UPLO = 'L', (in factorization order, k increases from 1 to N): a) A single positive entry IPIV(k) > 0 means: D(k,k) is a 1-by-1 diagonal block. If IPIV(k) != k, rows and columns k and IPIV(k) were interchanged in the matrix A(1:N,1:N). If IPIV(k) = k, no interchange occurred.

b) A pair of consecutive negative entries IPIV(k) < 0 and IPIV(k+1) < 0 means: D(k:k+1,k:k+1) is a 2-by-2 diagonal block. (NOTE: negative entries in IPIV appear ONLY in pairs). 1) If -IPIV(k) != k, rows and columns k and -IPIV(k) were interchanged in the matrix A(1:N,1:N). If -IPIV(k) = k, no interchange occurred. 2) If -IPIV(k+1) != k+1, rows and columns k-1 and -IPIV(k-1) were interchanged in the matrix A(1:N,1:N). If -IPIV(k+1) = k+1, no interchange occurred.

c) In both cases a) and b), always ABS(IPIV(k)) >= k.

d) NOTE: Any entry IPIV(k) is always NONZERO on output.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)).. On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >=1. For best performance LWORK >= N*NB, where NB is the block size returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit

< 0: If INFO = -k, the k-th argument had an illegal value

> 0: If INFO = k, the matrix A is singular, because: If UPLO = 'U': column k in the upper triangular part of A contains all zeros. If UPLO = 'L': column k in the lower triangular part of A contains all zeros.

Therefore D(k,k) is exactly zero, and superdiagonal elements of column k of U (or subdiagonal elements of column k of L) are all zeros. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

NOTE: INFO only stores the first occurrence of a singularity, any subsequent occurrence of singularity is not stored in INFO even though the factorization always completes.

Related Information

For this routine in other precisions, please see *chetrf_rk*. It also exists with a native C interface as *LAPACKE_zhetrf_rk*.

4.3.167 zhetrf_rook

ZHETRF_ROOK computes the factorization of a complex Hermitian matrix A using the bounded Bunch-Kaufman ("rook") diagonal pivoting method. The form of the factorization is

```
A = U*D*U**T  or  A = L*D*L**T
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is Hermitian and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhetrf_rook(UPLO, N, A, LDA, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhetrf_rook_(const char *uplo, const armpl_int_t *n,
                  armpl_doublecomplex_t *a, const armpl_int_t *lda,
                  armpl_int_t *ipiv, armpl_doublecomplex_t *work,
                  const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L (see below for further details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D.

If UPLO = 'U': Only the last KB elements of IPIV are set.

If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k-1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k-1 and -IPIV(k-1) were inerchaged, D(k-1:k,k-1:k) is a 2-by-2 diagonal block.

If UPLO = 'L': Only the first KB elements of IPIV are set.

If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k+1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k+1 and -IPIV(k+1) were inerchaged, D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)).. On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >=1. For best performance LWORK >= N*NB, where NB is the block size returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *chetrf_rook*. It also exists with a native C interface as *LAPACKE_zhetrf_rook*.

4.3.168 zhptrf

zhptrf computes the factorization of a complex Hermitian packed matrix A using the Bunch-Kaufman diagonal pivoting method:

```
A = U*D*U**H  or  A = L*D*L**H
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is Hermitian and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhptrf(UPLO, N, AP, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void zhptrf_(const char *uplo, const armpl_int_t *n,
             armpl_doublecomplex_t *ap, armpl_int_t *ipiv, armpl_int_t *info,
             ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the Hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L, stored as a packed triangular matrix overwriting A (see below for further details).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *chptrf*. It also exists with a native C interface as *LAPACKE_zhptrf*.

4.3.169 zpbtrf

zpbtrf computes the Cholesky factorization of a complex Hermitian positive definite band matrix A.

The factorization has the form

```
A = U**H * U,  if UPLO = 'U', or
A = L  * L**H,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is lower triangular.

Syntax

Fortran specification:

```
use armpl_library

subroutine zpbtrf(UPLO, N, KD, AB, LDAB, INFO)
```

C specification:

```
#include "armpl.h"

void zpbtrf_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
             armpl_doublecomplex_t *ab, const armpl_int_t *ldab,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the Hermitian band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB

as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, if INFO = 0, the triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H of the band matrix A, in the same storage format as A.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i is not positive definite, and the factorization could not be completed.

Related Information

For this routine in other precisions, please see *cpbtrf*, *dpbtrf* and *spbtrf*. It also exists with a native C interface as *LAPACKE_zpbtrf*.

4.3.170 zpftrf

zpftrf computes the Cholesky factorization of a complex Hermitian positive definite matrix A.

The factorization has the form

```
A = U**H * U,  if UPLO = 'U', or
A = L  * L**H,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is lower triangular.

This is the block version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine zpftrf(TRANSR, UPLO, N, A, INFO)
```

C specification:

```
#include "armpl.h"

void zpftrf_(const char *transr, const char *uplo, const armpl_int_t *n,
             armpl_doublecomplex_t *a, armpl_int_t *info, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': The Normal TRANSR of RFP A is stored; = 'C': The Conjugate-transpose TRANSR of RFP A is stored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of RFP A is stored; = 'L': Lower triangle of RFP A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (N*(N+1)/2);. On entry, the Hermitian matrix A in RFP format. RFP format is described by TRANSR, UPLO, and N as follows: If TRANSR = 'N' then RFP A is (0:N,0:k-1) when N is even; k=N/2. RFP A is (0:N-1,0:k) when N is odd; k=N/2. IF TRANSR = 'C' then RFP is the Conjugate-transpose of RFP A as defined when TRANSR = 'N'. The contents of RFP A are defined by UPLO as follows: If UPLO = 'U' the RFP A contains the nt elements of upper packed A. If UPLO = 'L' the RFP A contains the elements of lower packed A. The LDA of RFP A is (N+1)/2 when TRANSR = 'C'. When TRANSR is 'N' the LDA is N+1 when N is even and N is odd. See the Note below for more details.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization RFP A = U^H *U or RFP A = L*L^H .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i is not positive definite, and the factorization could not be completed.

Further Notes on RFP Format:

We first consider Standard Packed Format when N is even. We give an example where N = 6.

AP is Upper AP is Lower

00 01 02 03 04 05 00 11 12 13 14 15 10 11 22 23 24 25 20 21 22 33 34 35 30 31 32 33 44 45 40 41 42 43 44 55 50 51 52 53 54 55

Let TRANSR = 'N'. RFP holds AP as follows: For UPLO = 'U' the upper trapezoid A(0:5,0:2) consists of the last three columns of AP upper. The lower triangle A(4:6,0:2) consists of conjugate-transpose of the first three columns of AP upper. For UPLO = 'L' the lower trapezoid A(1:6,0:2) consists of the first three columns of AP lower. The upper triangle A(0:2,0:2) consists of conjugate-transpose of the last three columns of AP lower. To denote conjugate we place – above the element. This covers the case N even and TRANSR = 'N'.

RFP A RFP A

– – – 03 04 05 33 43 53 – – 13 14 15 00 44 54 – 23 24 25 10 11 55

33 34 35 20 21 22 – 00 44 45 30 31 32 – – 01 11 55 40 41 42 – – – 02 12 22 50 51 52

Now let TRANSR = 'C'. RFP A in both UPLO cases is just the conjugate- transpose of RFP A above. One therefore gets:

RFP A RFP A

– – – – – – – – – – 03 13 23 33 00 01 02 33 00 10 20 30 40 50 – – – – – – – – – – 04 14 24 34 44 11 12 43 44 11 21 31 41 51 – – – – – – – – – – 05 15 25 35 45 55 22 53 54 55 22 32 42 52

We next consider Standard Packed Format when N is odd. We give an example where N = 5.

AP is Upper AP is Lower

00 01 02 03 04 00 11 12 13 14 10 11 22 23 24 20 21 22 33 34 30 31 32 33 44 40 41 42 43 44

Let TRANSR = 'N'. RFP holds AP as follows: For UPLO = 'U' the upper trapezoid A(0:4,0:2) consists of the last three columns of AP upper. The lower triangle A(3:4,0:1) consists of conjugate-transpose of the first two columns of AP upper. For UPLO = 'L' the lower trapezoid A(0:4,0:2) consists of the first three columns of

AP lower. The upper triangle A(0:1,1:2) consists of conjugate-transpose of the last two columns of AP lower. To denote conjugate we place – above the element. This covers the case N odd and TRANSR = 'N'.

RFP A RFP A

– – 02 03 04 00 33 43 – 12 13 14 10 11 44

22 23 24 20 21 22 – 00 33 34 30 31 32 – – 01 11 44 40 41 42

Now let TRANSR = 'C'. RFP A in both UPLO cases is just the conjugate- transpose of RFP A above. One therefore gets:

RFP A RFP A

– – – – – – – – – 02 12 22 00 01 00 10 20 30 40 50 – – – – – – – – – 03 13 23 33 11 33 11 21 31 41 51 – – – – – – – – – – 04 14 24 34 44 43 44 22 32 42 52

Related Information

For this routine in other precisions, please see *cpftrf*, *dpftrf* and *spftrf*. It also exists with a native C interface as *LAPACKE_zpftrf*.

4.3.171 zpotrf

zpotrf computes the Cholesky factorization of a complex Hermitian positive definite matrix A.

The factorization has the form

```
A = U**H * U,  if UPLO = 'U', or
A = L  * L**H,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is lower triangular.

This is the block version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine zpotrf(UPLO, N, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void zpotrf_(const char *uplo, const armpl_int_t *n, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H .

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i is not positive definite, and the factorization could not be completed.

Related Information

For this routine in other precisions, please see *cpotrf*, *dpotrf* and *spotrf*. It also exists with a native C interface as *LAPACKE_zpotrf*.

4.3.172 zpotrf2

zpotrf2 computes the Cholesky factorization of a real symmetric positive definite matrix A using the recursive algorithm.

The factorization has the form

```
A = U**H * U,  if UPLO = 'U', or
A = L  * L**H,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is lower triangular.

This is the recursive version of the algorithm. It divides the matrix into four submatrices:

```
    [  A11 | A12  ]  where A11 is n1 by n1 and A22 is n2 by n2
A = [ -----|-----  ]  with n1 = n/2
    [  A21 | A22  ]       n2 = n-n1
```

The subroutine calls itself to factor A11. Update and scale A21 or A12, update A22 then call itself to factor A22.

Syntax

Fortran specification:

```
use armpl_library

recursive subroutine zpotrf2(UPLO, N, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void zpotrf2_(const char *uplo, const armpl_int_t *n,
              armpl_doublecomplex_t *a, const armpl_int_t *lda,
              armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H .

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i is not positive definite, and the factorization could not be completed.

Related Information

For this routine in other precisions, please see *cpotrf2*, *dpotrf2* and *spotrf2*. It also exists with a native C interface as *LAPACKE_zpotrf2*.

4.3.173 zpptrf

zpptrf computes the Cholesky factorization of a complex Hermitian positive definite matrix A stored in packed format.

The factorization has the form

```
A = U**H * U,  if UPLO = 'U', or
A = L  * L**H,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is lower triangular.

Syntax

Fortran specification:

```
use armpl_library

subroutine zpptrf(UPLO, N, AP, INFO)
```

C specification:

```
#include "armpl.h"

void zpptrf_(const char *uplo, const armpl_int_t *n,
             armpl_doublecomplex_t *ap, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the Hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n. See below for further details.

On exit, if INFO = 0, the triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H , in the same storage format as A.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i is not positive definite, and the factorization could not be completed.

Related Information

For this routine in other precisions, please see *cpptrf*, *dpptrf* and *spptrf*. It also exists with a native C interface as *LAPACKE_zpptrf*.

4.3.174 zpstrf

zpstrf computes the Cholesky factorization with complete pivoting of a complex Hermitian positive semidefinite matrix A.

The factorization has the form

```
P**T * A * P = U**H * U ,  if UPLO = 'U',
P**T * A * P = L  * L**H,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is lower triangular, and P is stored as vector PIV.

This algorithm does not attempt to check that A is positive semidefinite. This version of the algorithm calls level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine zpstrf(UPLO, N, A, LDA, PIV, RANK, TOL, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zpstrf_(const char *uplo, const armpl_int_t *n, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *piv, armpl_int_t *rank,
             const double *tol, double *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored. = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading n by n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n by n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization as above.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

PIV Output parameter.

PIV is INTEGER array, dimension (N)

PIV is such that the nonzero entries are P(PIV(K), K) = 1.

RANK Output parameter.

RANK is INTEGER

The rank of A given by the number of steps the algorithm completed.

TOL Input parameter.

TOL is DOUBLE PRECISION

User defined tolerance. If TOL < 0, then N*U*MAX(A(K,K)) will be used. The algorithm terminates at the (K-1)st step if the pivot <= TOL.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (2*N). Work space.

INFO Output parameter.

INFO is INTEGER

< 0: If INFO = -K, the K-th argument had an illegal value, = 0: algorithm completed successfully, and > 0: the matrix A is either rank deficient with computed rank as returned in RANK, or is not positive semidefinite. See Section 7 of LAPACK Working Note #161 for further information.

Related Information

For this routine in other precisions, please see *cpstrf*, *dpstrf* and *spstrf*. It also exists with a native C interface as *LAPACKE_zpstrf*.

4.3.175 zpttrf

zpttrf computes the L*D*L^H factorization of a complex Hermitian positive definite tridiagonal matrix A. The factorization may also be regarded as having the form A = U^H *D*U.

Syntax

Fortran specification:

```
use armpl_library

subroutine zpttrf(N, D, E, INFO)
```

C specification:

```
#include "armpl.h"

void zpttrf_(const armpl_int_t *n, double *d, armpl_doublecomplex_t *e,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry, the n diagonal elements of the tridiagonal matrix A. On exit, the n diagonal elements of the diagonal matrix D from the L*D*L^H factorization of A.

E Input and output parameter.

E is COMPLEX*16

E is an array, dimension (N-1). On entry, the (n-1) subdiagonal elements of the tridiagonal matrix A. On exit, the (n-1) subdiagonal elements of the unit bidiagonal factor L from the L*D*L^H factorization of A. E can also be regarded as the superdiagonal of the unit bidiagonal factor U from the U^H *D*U factorization of A.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value > 0: if INFO = k, the leading minor of order k is not positive definite; if k < N, the factorization could not be completed, while if k = N, the factorization was completed, but D(N) <= 0.

Related Information

For this routine in other precisions, please see *cpttrf*, *dpttrf* and *spttrf*. It also exists with a native C interface as *LAPACKE_zpttrf*.

4.3.176 zsptrf

zsptrf computes the factorization of a complex symmetric matrix A stored in packed format using the Bunch-Kaufman diagonal pivoting method:

```
A = U*D*U**T  or  A = L*D*L**T
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsptrf(UPLO, N, AP, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void zsptrf_(const char *uplo, const armpl_int_t *n,
             armpl_doublecomplex_t *ap, armpl_int_t *ipiv, armpl_int_t *info,
             ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L, stored as a packed triangular matrix overwriting A (see below for further details).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *csptrf*, *dsptrf* and *ssptrf*. It also exists with a native C interface as *LAPACKE_zsptrf*.

4.3.177 zsytrf

zsytrf computes the factorization of a complex symmetric matrix A using the Bunch-Kaufman diagonal pivoting method. The form of the factorization is

```
A = U*D*U**T  or  A = L*D*L**T
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is symmetric and block diagonal with with 1-by-1 and 2-by-2 diagonal blocks.

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsytrf(UPLO, N, A, LDA, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zsytrf_(const char *uplo, const armpl_int_t *n, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *ipiv,
             armpl_doublecomplex_t *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L (see below for further details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >=1. For best performance LWORK >= N*NB, where NB is the block size returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *csytrf*, *dsytrf* and *ssytrf*. It also exists with a native C interface as *LAPACKE_zsytrf*.

4.3.178 zsytrf_aa

ZSYTRF_AA computes the factorization of a complex symmetric matrix A using the Aasen's algorithm. The form of the factorization is

```
A = U*T*U**T  or  A = L*T*L**T
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and T is a complex symmetric tridiagonal matrix.

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsytrf_aa(UPLO, N, A, LDA, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zsytrf_aa_(const char *uplo, const armpl_int_t *n,
                armpl_doublecomplex_t *a, const armpl_int_t *lda,
                armpl_int_t *ipiv, armpl_doublecomplex_t *work,
                const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, the tridiagonal matrix is stored in the diagonals and the subdiagonals of A just below (or above) the diagonals, and L is stored below (or above) the subdiaonals, when UPLO is 'L' (or 'U').

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

On exit, it contains the details of the interchanges, i.e., the row and column k of A were interchanged with the row and column IPIV(k).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >=MAX(1,2*N). For optimum performance LWORK >= N*(1+NB), where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *csytrf_aa*, *dsytrf_aa* and *ssytrf_aa*. It also exists with a native C interface as *LAPACKE_zsytrf_aa*.

4.3.179 zsytrf_rk

ZSYTRF_RK computes the factorization of a complex symmetric matrix A using the bounded Bunch-Kaufman (rook) diagonal pivoting method:

```
A = P*U*D*(U**T)*(P**T) or A = P*L*D*(L**T)*(P**T),
```

where U (or L) is unit upper (or lower) triangular matrix, U^T (or L^T) is the transpose of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the blocked version of the algorithm, calling Level 3 BLAS. For more information see Further Details section.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsytrf_rk(UPLO, N, A, LDA, E, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zsytrf_rk_(const char *uplo, const armpl_int_t *n,
                armpl_doublecomplex_t *a, const armpl_int_t *lda,
                armpl_doublecomplex_t *e, armpl_int_t *ipiv,
                armpl_doublecomplex_t *work, const armpl_int_t *lwork,
                armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U': the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced.

If UPLO = 'L': the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, contains: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Output parameter.

E is COMPLEX*16

E is an array, dimension (N). On exit, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) is set to 0; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) is set to 0.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is set to 0 in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

IPIV describes the permutation matrix P in the factorization of matrix A as follows. The absolute value of IPIV(k) represents the index of row and column that were interchanged with the k-th row and column. The value of UPLO describes the order in which the interchanges were applied. Also, the sign of IPIV represents the block structure of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks which correspond to 1 or 2 interchanges at each factorization step. For more info see Further Details section.

If UPLO = 'U', (in factorization order, k decreases from N to 1): a) A single positive entry IPIV(k) > 0 means: D(k,k) is a 1-by-1 diagonal block. If IPIV(k) != k, rows and columns k and IPIV(k) were interchanged in the matrix A(1:N,1:N); If IPIV(k) = k, no interchange occurred.

b) A pair of consecutive negative entries IPIV(k) < 0 and IPIV(k-1) < 0 means: D(k-1:k,k-1:k) is a 2-by-2 diagonal block. (NOTE: negative entries in IPIV appear ONLY in pairs). 1) If -IPIV(k) != k, rows and columns k and -IPIV(k) were interchanged in the matrix A(1:N,1:N). If -IPIV(k) = k, no interchange occurred. 2) If -IPIV(k-1) != k-1, rows and columns k-1 and -IPIV(k-1) were interchanged in the matrix A(1:N,1:N). If -IPIV(k-1) = k-1, no interchange occurred.

c) In both cases a) and b), always ABS(IPIV(k)) <= k.

d) NOTE: Any entry IPIV(k) is always NONZERO on output.

If UPLO = 'L', (in factorization order, k increases from 1 to N): a) A single positive entry IPIV(k) > 0 means: D(k,k) is a 1-by-1 diagonal block. If IPIV(k) != k, rows and columns k and IPIV(k) were interchanged in the matrix A(1:N,1:N). If IPIV(k) = k, no interchange occurred.

b) A pair of consecutive negative entries IPIV(k) < 0 and IPIV(k+1) < 0 means: D(k:k+1,k:k+1) is a 2-by-2 diagonal block. (NOTE: negative entries in IPIV appear ONLY in pairs). 1) If -IPIV(k) != k, rows and columns k and -IPIV(k) were interchanged in the matrix A(1:N,1:N). If -IPIV(k) = k, no interchange occurred. 2) If -IPIV(k+1) != k+1, rows and columns k-1 and -IPIV(k-1) were interchanged in the matrix A(1:N,1:N). If -IPIV(k+1) = k+1, no interchange occurred.

c) In both cases a) and b), always ABS(IPIV(k)) >= k.

d) NOTE: Any entry IPIV(k) is always NONZERO on output.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)).. On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >=1. For best performance LWORK >= N*NB, where NB is the block size returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit

< 0: If INFO = -k, the k-th argument had an illegal value

> 0: If INFO = k, the matrix A is singular, because: If UPLO = 'U': column k in the upper triangular part of A contains all zeros. If UPLO = 'L': column k in the lower triangular part of A contains all zeros.

Therefore D(k,k) is exactly zero, and superdiagonal elements of column k of U (or subdiagonal elements of column k of L) are all zeros. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

NOTE: INFO only stores the first occurrence of a singularity, any subsequent occurrence of singularity is not stored in INFO even though the factorization always completes.

Related Information

For this routine in other precisions, please see *csytrf_rk*, *dsytrf_rk* and *ssytrf_rk*. It also exists with a native C interface as *LAPACKE_zsytrf_rk*.

4.3.180 zsytrf_rook

ZSYTRF_ROOK computes the factorization of a complex symmetric matrix A using the bounded Bunch-Kaufman ("rook") diagonal pivoting method. The form of the factorization is

```
A = U*D*U**T  or  A = L*D*L**T
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsytrf_rook(UPLO, N, A, LDA, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zsytrf_rook_(const char *uplo, const armpl_int_t *n,
                  armpl_doublecomplex_t *a, const armpl_int_t *lda,
                  armpl_int_t *ipiv, armpl_doublecomplex_t *work,
                  const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L (see below for further details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D.

If UPLO = 'U': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k-1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k-1 and -IPIV(k-1) were inerchaged, D(k-1:k,k-1:k) is a 2-by-2 diagonal block.

If UPLO = 'L': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k+1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k+1 and -IPIV(k+1) were inerchaged, D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)).. On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >=1. For best performance LWORK >= N*NB, where NB is the block size returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *csytrf_rook*, *dsytrf_rook* and *ssytrf_rook*. It also exists with a native C interface as *LAPACKE_zsytrf_rook*.

4.3.181 ztplqt

DTPLQT computes a blocked LQ factorization of a complex "triangular-pentagonal" matrix C, which is composed of a triangular block A and pentagonal block B, using the compact WY representation for Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine ztplqt(M, N, L, MB, A, LDA, B, LDB, T, LDT, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void ztplqt_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *l,
             const armpl_int_t *mb, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_doublecomplex_t *b,
```

(continues on next page)

(continued from previous page)

```
              const armpl_int_t *ldb, armpl_doublecomplex_t *t,
              const armpl_int_t *ldt, armpl_doublecomplex_t *work,
              armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix B, and the order of the triangular matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix B. N >= 0.

L Input parameter.

L is INTEGER

The number of rows of the lower trapezoidal part of B. MIN(M, N) >= L >= 0. See Further Details.

MB Input parameter.

MB is INTEGER

The block size to be used in the blocked QR. M >= MB >= 1.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, M). On entry, the lower triangular M-by-M matrix A. On exit, the elements on and below the diagonal of the array contain the lower triangular matrix L.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). On entry, the pentagonal M-by-N matrix B. The first N-L columns are rectangular, and the last L columns are lower trapezoidal. On exit, B contains the pentagonal matrix V. See Further Details.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M).

T Output parameter.

T is COMPLEX*16

T is an array, dimension (LDT, N). The lower triangular block reflectors stored in compact form as a sequence of upper triangular blocks. See Further Details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= MB.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MB*M) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctplqt*, *dtplqt* and *stplqt*.

4.3.182 ztpmlqt

ztpmlqt applies a complex orthogonal matrix Q obtained from a "triangular-pentagonal" complex block reflector H to a general complex matrix C, which consists of two blocks A and B.

Syntax

Fortran specification:

```
use armpl_library

subroutine ztpmlqt(SIDE, TRANS, M, N, K, L, MB, V, LDV, T, LDT, A, LDA, B,
                   LDB, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void ztpmlqt_(const char *side, const char *trans, const armpl_int_t *m,
              const armpl_int_t *n, const armpl_int_t *k,
              const armpl_int_t *l, const armpl_int_t *mb,
              const armpl_doublecomplex_t *v, const armpl_int_t *ldv,
              const armpl_doublecomplex_t *t, const armpl_int_t *ldt,
              armpl_doublecomplex_t *a, const armpl_int_t *lda,
              armpl_doublecomplex_t *b, const armpl_int_t *ldb,
              armpl_doublecomplex_t *work, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left; = 'R': apply Q or Q^H from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'C': Transpose, apply Q^H .

M Input parameter.

M is INTEGER

The number of rows of the matrix B. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix B. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q.

L Input parameter.

L is INTEGER

The order of the trapezoidal part of V. K >= L >= 0. See Further Details.

MB Input parameter.

MB is INTEGER

The block size used for the storage of T. K >= MB >= 1. This must be the same value of MB used to generate T in DTPLQT.

V Input parameter.

V is COMPLEX*16

V is an array, dimension (LDA, K). The i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by DTPLQT in B. See Further Details.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. If SIDE = 'L', LDV >= max(1, M); if SIDE = 'R', LDV >= max(1, N).

T Input parameter.

T is COMPLEX*16

T is an array, dimension (LDT, K). The upper triangular factors of the block reflectors as returned by DTPLQT, stored as a MB-by-K matrix.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= MB.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension. (LDA, N) if SIDE = 'L' or (LDA, K) if SIDE = 'R' On entry, the K-by-N or M-by-K matrix A. On exit, A is overwritten by the corresponding block of Q*C or Q^H *C or C*Q or C*Q^H . See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDC >= max(1, K); If SIDE = 'R', LDC >= max(1, M).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). On entry, the M-by-N matrix B. On exit, B is overwritten by the corresponding block of Q*C or Q^H *C or C*Q or C*Q^H . See Further Details.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M).

WORK Output parameter.

WORK is COMPLEX*16 array. The dimension of WORK is

N* MB if SIDE = 'L', or M* MB if SIDE = 'R'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctpmlqt*, *dtpmlqt* and *stpmlqt*.

4.3.183 ztpmqrt

ztpmqrt applies a complex orthogonal matrix Q obtained from a "triangular-pentagonal" complex block reflector H to a general complex matrix C, which consists of two blocks A and B.

Syntax

Fortran specification:

```
use armpl_library

subroutine ztpmqrt(SIDE, TRANS, M, N, K, L, NB, V, LDV, T, LDT, A, LDA, B,
                   LDB, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void ztpmqrt_(const char *side, const char *trans, const armpl_int_t *m,
              const armpl_int_t *n, const armpl_int_t *k,
              const armpl_int_t *l, const armpl_int_t *nb,
              const armpl_doublecomplex_t *v, const armpl_int_t *ldv,
              const armpl_doublecomplex_t *t, const armpl_int_t *ldt,
              armpl_doublecomplex_t *a, const armpl_int_t *lda,
              armpl_doublecomplex_t *b, const armpl_int_t *ldb,
              armpl_doublecomplex_t *work, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left; = 'R': apply Q or Q^H from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'C': Transpose, apply Q^H .

M Input parameter.

M is INTEGER

The number of rows of the matrix B. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix B. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q.

L Input parameter.

L is INTEGER

The order of the trapezoidal part of V. K >= L >= 0. See Further Details.

NB Input parameter.

NB is INTEGER

The block size used for the storage of T. K >= NB >= 1. This must be the same value of NB used to generate T in CTPQRT.

V Input parameter.

V is COMPLEX*16

V is an array, dimension (LDA, K). The i-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by CTPQRT in B. See Further Details.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. If SIDE = 'L', LDV >= max(1, M); if SIDE = 'R', LDV >= max(1, N).

T Input parameter.

T is COMPLEX*16

T is an array, dimension (LDT, K). The upper triangular factors of the block reflectors as returned by CTPQRT, stored as a NB-by-K matrix.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= NB.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension. (LDA, N) if SIDE = 'L' or (LDA, K) if SIDE = 'R' On entry, the K-by-N or M-by-K matrix A. On exit, A is overwritten by the corresponding block of Q*C or Q^H *C or C*Q or C*Q^H . See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDC >= max(1, K); If SIDE = 'R', LDC >= max(1, M).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). On entry, the M-by-N matrix B. On exit, B is overwritten by the corresponding block of Q*C or Q^H *C or C*Q or C*Q^H . See Further Details.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M).

WORK Output parameter.

WORK is COMPLEX*16 array. The dimension of WORK is

N* NB if SIDE = 'L', or M* NB if SIDE = 'R'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctpmqrt*, *dtpmqrt* and *stpmqrt*. It also exists with a native C interface as *LAPACKE_ztpmqrt*.

4.3.184 ztpqrt

ztpqrt computes a blocked QR factorization of a complex "triangular-pentagonal" matrix C, which is composed of a triangular block A and pentagonal block B, using the compact WY representation for Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine ztpqrt(M, N, L, NB, A, LDA, B, LDB, T, LDT, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void ztpqrt_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *l,
             const armpl_int_t *nb, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, armpl_doublecomplex_t *t,
             const armpl_int_t *ldt, armpl_doublecomplex_t *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix B. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix B, and the order of the triangular matrix A. N >= 0.

L Input parameter.

L is INTEGER

The number of rows of the upper trapezoidal part of B. MIN(M, N) >= L >= 0. See Further Details.

NB Input parameter.

NB is INTEGER

The block size to be used in the blocked QR. N >= NB >= 1.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the upper triangular N-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the upper triangular matrix R.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). On entry, the pentagonal M-by-N matrix B. The first M-L rows are rectangular, and the last L rows are upper trapezoidal. On exit, B contains the pentagonal matrix V. See Further Details.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M).

T Output parameter.

T is COMPLEX*16

T is an array, dimension (LDT, N). The upper triangular block reflectors stored in compact form as a sequence of upper triangular blocks. See Further Details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= NB.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (NB*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctpqrt*, *dtpqrt* and *stpqrt*. It also exists with a native C interface as *LAPACKE_ztpqrt*.

4.3.185 ztzrzf

`ztzrzf` reduces the M-by-N (M<=N) complex upper trapezoidal matrix A to upper triangular form by means of unitary transformations.

The upper trapezoidal matrix A is factored as

```
A = ( R  0 ) * Z,
```

where Z is an N-by-N unitary matrix and R is an M-by-M upper triangular matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine ztzrzf(M, N, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ztzrzf_(const armpl_int_t *m, const armpl_int_t *n,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_doublecomplex_t *tau, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

`M` is `INTEGER`

The number of rows of the matrix `A`. `M` >= 0.

N Input parameter.

`N` is `INTEGER`

The number of columns of the matrix `A`. `N` >= `M`.

A Input and output parameter.

`A` is `COMPLEX*16`

`A` is an array, dimension (`LDA`, `N`). On entry, the leading M-by-N upper trapezoidal part of the array `A` must contain the matrix to be factorized. On exit, the leading M-by-M upper triangular part of A contains the upper triangular matrix R, and elements M+1 to `N` of the first `M` rows of `A`, with the array `TAU`, represent the unitary matrix Z as a product of `M` elementary reflectors.

LDA Input parameter.

`LDA` is `INTEGER`

The leading dimension of the array `A`. `LDA` >= max(1, `M`).

TAU Output parameter.

`TAU` is `COMPLEX*16`

`TAU` is an array, dimension (`M`). The scalar factors of the elementary reflectors.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, M). For optimum performance LWORK >= M*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctzrzf*, *dtzrzf* and *stzrzf*. It also exists with a native C interface as *LAPACKE_ztzrzf*.

4.3.186 zunglq

zunglq generates an M-by-N complex matrix Q with orthonormal rows, which is defined as the first M rows of a product of K elementary reflectors of order N

```
Q  =  H(k)**H . . . H(2)**H H(1)**H
```

as returned by ZGELQF.

Syntax

Fortran specification:

```
use armpl_library

subroutine zunglq(M, N, K, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zunglq_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *k,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_doublecomplex_t *tau, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix Q. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix Q. N >= M.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. M >= K >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by ZGELQF in the first k rows of its array argument A. On exit, the M-by-N matrix Q.

LDA Input parameter.

LDA is INTEGER

The first dimension of the array A. LDA >= max(1, M).

TAU Input parameter.

TAU is COMPLEX*16

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by ZGELQF.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, M). For optimum performance LWORK >= M*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit; < 0: if INFO = -i, the i-th argument has an illegal value

Related Information

For this routine in other precisions, please see *cunglq*. It also exists with a native C interface as *LAPACKE_zunglq*.

4.3.187 zungql

zungql generates an M-by-N complex matrix Q with orthonormal columns, which is defined as the last N columns of a product of K elementary reflectors of order M

```
Q  =  H(k) . . . H(2) H(1)
```

as returned by ZGEQLF.

Syntax

Fortran specification:

```
use armpl_library

subroutine zungql(M, N, K, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zungql_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *k,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_doublecomplex_t *tau, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix Q. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix Q. M >= N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. N >= K >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the (n-k+i)-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by ZGEQLF in the last k columns of its array argument A. On exit, the M-by-N matrix Q.

LDA Input parameter.

LDA is INTEGER

The first dimension of the array A. LDA >= max(1, M).

TAU Input parameter.

TAU is COMPLEX*16

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by ZGEQLF.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N). For optimum performance LWORK >= N*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument has an illegal value

Related Information

For this routine in other precisions, please see *cungql*. It also exists with a native C interface as *LAPACKE_zungql*.

4.3.188 zungqr

zungqr generates an M-by-N complex matrix Q with orthonormal columns, which is defined as the first N columns of a product of K elementary reflectors of order M

```
Q  =  H(1) H(2) . . . H(k)
```

as returned by ZGEQRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine zungqr(M, N, K, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zungqr_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *k,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_doublecomplex_t *tau, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix Q. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix Q. M >= N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. N >= K >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the i-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by ZGEQRF in the first k columns of its array argument A. On exit, the M-by-N matrix Q.

LDA Input parameter.

LDA is INTEGER

The first dimension of the array A. LDA >= max(1, M).

TAU Input parameter.

TAU is COMPLEX*16

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by ZGEQRF.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N). For optimum performance LWORK >= N*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument has an illegal value

Related Information

For this routine in other precisions, please see *cungqr*. It also exists with a native C interface as *LAPACKE_zungqr*.

4.3.189 zungrq

zungrq generates an M-by-N complex matrix Q with orthonormal rows, which is defined as the last M rows of a product of K elementary reflectors of order N

```
Q  =  H(1)**H H(2)**H . . . H(k)**H
```

as returned by ZGERQF.

Syntax

Fortran specification:

```
use armpl_library

subroutine zungrq(M, N, K, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zungrq_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *k,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_doublecomplex_t *tau, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix Q. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix Q. N >= M.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. M >= K >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the (m-k+i)-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by ZGERQF in the last k rows of its array argument A. On exit, the M-by-N matrix Q.

LDA Input parameter.

LDA is INTEGER

The first dimension of the array A. LDA >= max(1, M).

TAU Input parameter.

TAU is COMPLEX*16

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by ZGERQF.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, M). For optimum performance LWORK >= M*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument has an illegal value

Related Information

For this routine in other precisions, please see *cungrq*. It also exists with a native C interface as *LAPACKE_zungrq*.

4.3.190 zunmlq

zunmlq overwrites the general complex M-by-N matrix C with

```
SIDE = 'L'      SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'C': Q^H * C C * Q^H

where Q is a complex unitary matrix defined as the product of k elementary reflectors

```
Q = H(k)**H . . . H(2)**H H(1)**H
```

as returned by ZGELQF. Q is of order M if SIDE = 'L' and of order N if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine zunmlq(SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void zunmlq_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_doublecomplex_t *tau, armpl_doublecomplex_t *c,
             const armpl_int_t *ldc, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left; = 'R': apply Q or Q^H from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'C': Conjugate transpose, apply Q^H .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension. (LDA, M) if SIDE = 'L', (LDA, N) if SIDE = 'R' The i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by ZGELQF in the first k rows of its array argument A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, K).

TAU Input parameter.

TAU is COMPLEX*16

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by ZGELQF.

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^H * C or C*Q^H or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, N); if SIDE = 'R', LWORK >= max(1, M). For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cunmlq*. It also exists with a native C interface as *LAPACKE_zunmlq*.

4.3.191 zunmql

zunmql overwrites the general complex M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'C': Q^H * C C * Q^H

where Q is a complex unitary matrix defined as the product of k elementary reflectors

```
Q = H(k) . . . H(2) H(1)
```

as returned by ZGEQLF. Q is of order M if SIDE = 'L' and of order N if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine zunmql(SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void zunmql_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_doublecomplex_t *tau, armpl_doublecomplex_t *c,
             const armpl_int_t *ldc, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left; = 'R': apply Q or Q^H from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'C': Transpose, apply Q^H .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, K). The i-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by ZGEQLF in the last k columns of its array argument A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDA >= max(1, M); if SIDE = 'R', LDA >= max(1, N).

TAU Input parameter.

TAU is COMPLEX*16

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by ZGEQLF.

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^H * C or C*Q^H or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, N); if SIDE = 'R', LWORK >= max(1, M). For good performance, LWORK should genreally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cunmql*. It also exists with a native C interface as *LAPACKE_zunmql*.

4.3.192 zunmqr

zunmqr overwrites the general complex M-by-N matrix C with

```
SIDE = 'L'      SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'C': Q^H * C C * Q^H

where Q is a complex unitary matrix defined as the product of k elementary reflectors

```
Q = H(1) H(2) . . . H(k)
```

as returned by ZGEQRF. Q is of order M if SIDE = 'L' and of order N if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine zunmqr(SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void zunmqr_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_doublecomplex_t *tau, armpl_doublecomplex_t *c,
             const armpl_int_t *ldc, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left; = 'R': apply Q or Q^H from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'C': Conjugate transpose, apply Q^H .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, K). The i-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by ZGEQRF in the first k columns of its array argument A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDA >= max(1, M); if SIDE = 'R', LDA >= max(1, N).

TAU Input parameter.

TAU is COMPLEX*16

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by ZGEQRF.

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^H * C or C*Q^H or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, N); if SIDE = 'R', LWORK >= max(1, M). For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cunmqr*. It also exists with a native C interface as *LAPACKE_zunmqr*.

4.3.193 zunmrq

zunmrq overwrites the general complex M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'C': Q^H * C C * Q^H

where Q is a complex unitary matrix defined as the product of k elementary reflectors

```
Q = H(1)**H H(2)**H . . . H(k)**H
```

as returned by ZGERQF. Q is of order M if SIDE = 'L' and of order N if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine zunmrq(SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void zunmrq_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_doublecomplex_t *tau, armpl_doublecomplex_t *c,
             const armpl_int_t *ldc, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left; = 'R': apply Q or Q^H from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'C': Transpose, apply Q^H .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension. (LDA, M) if SIDE = 'L', (LDA, N) if SIDE = 'R' The i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by ZGERQF in the last k rows of its array argument A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, K).

TAU Input parameter.

TAU is COMPLEX*16

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by ZGERQF.

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^H * C or C*Q^H or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, N); if SIDE = 'R', LWORK >= max(1, M). For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cunmrq*. It also exists with a native C interface as *LAPACKE_zunmrq*.

4.3.194 zunmrz

zunmrz overwrites the general complex M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'C': Q^H * C C * Q^H

where Q is a complex unitary matrix defined as the product of k elementary reflectors

```
Q = H(1) H(2) . . . H(k)
```

as returned by ZTZRZF. Q is of order M if SIDE = 'L' and of order N if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine zunmrz(SIDE, TRANS, M, N, K, L, A, LDA, TAU, C, LDC, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void zunmrz_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k, const armpl_int_t *l,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_doublecomplex_t *tau, armpl_doublecomplex_t *c,
             const armpl_int_t *ldc, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left; = 'R': apply Q or Q^H from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'C': Conjugate transpose, apply Q^H .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

L Input parameter.

L is INTEGER

The number of columns of the matrix A containing the meaningful part of the Householder reflectors. If SIDE = 'L', M >= L >= 0, if SIDE = 'R', N >= L >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension. (LDA, M) if SIDE = 'L', (LDA, N) if SIDE = 'R' The i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by ZTZRZF in the last k rows of its array argument A. A is modified by the routine but restored on exit.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, K).

TAU Input parameter.

TAU is COMPLEX*16

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by ZTZRZF.

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^H * C or C*Q^H or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, N); if SIDE = 'R', LWORK >= max(1, M). For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cunmrz*. It also exists with a native C interface as *LAPACKE_zunmrz*.

4.4 LAPACK matrix inversion routines

4.4.1 cgetri

cgetri computes the inverse of a matrix using the LU factorization computed by CGETRF.

This method inverts U and then computes inv(A) by solving the system inv(A)*L = inv(U) for inv(A).

Syntax

Fortran specification:

```
use armpl_library

subroutine cgetri(N, A, LDA, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgetri_(const armpl_int_t *n, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, const armpl_int_t *ipiv,
             armpl_singlecomplex_t *work, const armpl_int_t *lwork,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the factors L and U from the factorization A = P*L*U as computed by CGETRF. On exit, if INFO = 0, the inverse of the original matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from CGETRF; for 1<=i<=N, row i of the matrix was interchanged with row IPIV(i).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO=0, then WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N). For optimal performance LWORK >= N*NB, where NB is the optimal blocksize returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, U(i,i) is exactly zero; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *dgetri*, *sgetri* and *zgetri*. It also exists with a native C interface as *LAPACKE_cgetri*.

4.4.2 chetri

chetri computes the inverse of a complex Hermitian indefinite matrix A using the factorization A = U*D*U^H or A = L*D*L^H computed by CHETRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine chetri(UPLO, N, A, LDA, IPIV, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void chetri_(const char *uplo, const armpl_int_t *n, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, const armpl_int_t *ipiv,
             armpl_singlecomplex_t *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^H ; = 'L': Lower triangular, form is A = L*D*L^H .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CHETRF.

On exit, if INFO = 0, the (Hermitian) inverse of the original matrix. If UPLO = 'U', the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; if UPLO = 'L' the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CHETRF.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *zhetri*. It also exists with a native C interface as *LAPACKE_chetri*.

4.4.3 chetri2

chetri2 computes the inverse of a COMPLEX hermitian indefinite matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by CHETRF. chetri2 set the LEADING DIMENSION of the workspace before calling ‘‘chetri2‘‘X that actually computes the inverse.

Syntax

Fortran specification:

```
use armpl_library

subroutine chetri2(UPLO, N, A, LDA, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void chetri2_(const char *uplo, const armpl_int_t *n,
              armpl_singlecomplex_t *a, const armpl_int_t *lda,
              const armpl_int_t *ipiv, armpl_singlecomplex_t *work,
              const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = ‘U’: Upper triangular, form is A = U*D*U^T ; = ‘L’: Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the NB diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CHETRF.

On exit, if INFO = 0, the (symmetric) inverse of the original matrix. If UPLO = 'U', the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; if UPLO = 'L' the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the NB structure of D as determined by CHETRF.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (N+NB+1)*(NB+3) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. WORK is size >= (N+NB+1)*(NB+3) If LWORK = -1, then a workspace query is assumed; the routine calculates: - the optimal size of the WORK array, returns this value as the first entry of the WORK array, - and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *zhetri2*. It also exists with a native C interface as *LAPACKE_chetri2*.

4.4.4 chetri2x

CHETRI2X computes the inverse of a complex Hermitian indefinite matrix A using the factorization A = U*D*U^H or A = L*D*L^H computed by CHETRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine chetri2x(UPLO, N, A, LDA, IPIV, WORK, NB, INFO)
```

C specification:

```
#include "armpl.h"

void chetri2x_(const char *uplo, const armpl_int_t *n,
               armpl_singlecomplex_t *a, const armpl_int_t *lda,
               const armpl_int_t *ipiv, armpl_singlecomplex_t *work,
               const armpl_int_t *nb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^H ; = 'L': Lower triangular, form is A = L*D*L^H .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the NNB diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CHETRF.

On exit, if INFO = 0, the (symmetric) inverse of the original matrix. If UPLO = 'U', the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; if UPLO = 'L' the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the NNB structure of D as determined by CHETRF.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (N+NB+1,NB+3) .

NB Input parameter.

NB is INTEGER

Block size

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *zhetri2x*. It also exists with a native C interface as *LAPACKE_chetri2x*.

4.4.5 chetri_3

CHETRI_3 computes the inverse of a complex Hermitian indefinite matrix A using the factorization computed by CHETRF_RK or CHETRF_BK:

```
A = P*U*D*(U**H)*(P**T) or A = P*L*D*(L**H)*(P**T),
```

where U (or L) is unit upper (or lower) triangular matrix, U^H (or L^H) is the conjugate of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is Hermitian and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

CHETRI_3 sets the leading dimension of the workspace before calling CHETRI_3X that actually computes the inverse. This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine chetri_3(UPLO, N, A, LDA, E, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void chetri_3_(const char *uplo, const armpl_int_t *n,
               armpl_singlecomplex_t *a, const armpl_int_t *lda,
               const armpl_singlecomplex_t *e, const armpl_int_t *ipiv,
               armpl_singlecomplex_t *work, const armpl_int_t *lwork,
               armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, diagonal of the block diagonal matrix D and factors U or L as computed by CHETRF_RK and CHETRF_BK: a) ONLY diagonal elements of the Hermitian block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D should be provided on entry in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

On exit, if INFO = 0, the Hermitian inverse of the original matrix. If UPLO = 'U': the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; If UPLO = 'L': the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Input parameter.

E is COMPLEX

E is an array, dimension (N). On entry, contains the superdiagonal (or subdiagonal) elements of the Hermitian block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i),i=2:N, E(1) not referenced; If UPLO = 'L': E(i) = D(i+1,i),i=1:N-1, E(N) not referenced.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is not referenced in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CHETRF_RK or CHETRF_BK.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (N+NB+1)*(NB+3).. On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= (N+NB+1)*(NB+3).

If LDWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *zhetri_3*. It also exists with a native C interface as *LAPACKE_chetri_3*.

4.4.6 chetri_rook

CHETRI_ROOK computes the inverse of a complex Hermitian indefinite matrix A using the factorization A = $U*D*U^H$ or A = $L*D*L^H$ computed by CHETRF_ROOK.

Syntax

Fortran specification:

```
use armpl_library

subroutine chetri_rook(UPLO, N, A, LDA, IPIV, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void chetri_rook_(const char *uplo, const armpl_int_t *n,
                  armpl_singlecomplex_t *a, const armpl_int_t *lda,
                  const armpl_int_t *ipiv, armpl_singlecomplex_t *work,
                  armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^H ; = 'L': Lower triangular, form is A = L*D*L^H .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CHETRF_ROOK.

On exit, if INFO = 0, the (Hermitian) inverse of the original matrix. If UPLO = 'U', the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; if UPLO = 'L' the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CHETRF_ROOK.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *zhetri_rook*.

4.4.7 chptri

chptri computes the inverse of a complex Hermitian indefinite matrix A in packed storage using the factorization A = U*D*U^H or A = L*D*L^H computed by CHPTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine chptri(UPLO, N, AP, IPIV, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void chptri_(const char *uplo, const armpl_int_t *n,
             armpl_singlecomplex_t *ap, const armpl_int_t *ipiv,
             armpl_singlecomplex_t *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^H ; = 'L': Lower triangular, form is A = L*D*L^H .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). On entry, the block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CHPTRF, stored as a packed triangular matrix.

On exit, if INFO = 0, the (Hermitian) inverse of the original matrix, stored as a packed triangular matrix. The j-th column of inv(A) is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = inv(A)(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = inv(A)(i,j) for j<=i<=n.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CHPTRF.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *zhptri*. It also exists with a native C interface as *LAPACKE_chptri*.

4.4.8 cpftri

`cpftri` computes the inverse of a complex Hermitian positive definite matrix A using the Cholesky factorization $A = U^H * U$ or $A = L * L^H$ computed by CPFTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine cpftri(TRANSR, UPLO, N, A, INFO)
```

C specification:

```
#include "armpl.h"

void cpftri_(const char *transr, const char *uplo, const armpl_int_t *n,
             armpl_singlecomplex_t *a, armpl_int_t *info, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': The Normal TRANSR of RFP A is stored; = 'C': The Conjugate-transpose TRANSR of RFP A is stored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (N*(N+1)/2);. On entry, the Hermitian matrix A in RFP format. RFP format is described by TRANSR, UPLO, and N as follows: If TRANSR = 'N' then RFP A is (0:N,0:k-1) when N is even; k=N/2. RFP A is (0:N-1,0:k) when N is odd; k=N/2. IF TRANSR = 'C' then RFP is the Conjugate-transpose of RFP A as defined when TRANSR = 'N'. The contents of RFP A are defined by UPLO as follows: If UPLO = 'U' the RFP A contains the nt elements of upper packed A. If UPLO = 'L' the RFP A contains the elements of lower packed A. The LDA of RFP A is (N+1)/2 when TRANSR = 'C'. When TRANSR is 'N' the LDA is N+1 when N is even and N is odd. See the Note below for more details.

On exit, the Hermitian inverse of the original matrix, in the same storage format.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the (i,i) element of the factor U or L is zero, and the inverse could not be computed.

Related Information

For this routine in other precisions, please see *dpftri*, *spftri* and *zpftri*. It also exists with a native C interface as *LAPACKE_cpftri*.

4.4.9 cpotri

`cpotri` computes the inverse of a complex Hermitian positive definite matrix A using the Cholesky factorization A = U^H *U or A = L*L^H computed by CPOTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine cpotri(UPLO, N, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void cpotri_(const char *uplo, const armpl_int_t *n, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H , as computed by CPOTRF. On exit, the upper or lower triangle of the (Hermitian) inverse of A, overwriting the input factor U or L.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the (i,i) element of the factor U or L is zero, and the inverse could not be computed.

Related Information

For this routine in other precisions, please see *dpotri*, *spotri* and *zpotri*. It also exists with a native C interface as *LAPACKE_cpotri*.

4.4.10 cpptri

`cpptri` computes the inverse of a complex Hermitian positive definite matrix A using the Cholesky factorization $A = U^H * U$ or $A = L * L^H$ computed by CPPTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine cpptri(UPLO, N, AP, INFO)
```

C specification:

```
#include "armpl.h"

void cpptri_(const char *uplo, const armpl_int_t *n,
             armpl_singlecomplex_t *ap, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

= 'U': Upper triangular factor is stored in AP; = 'L': Lower triangular factor is stored in `AP`.

N Input parameter.

`N` is `INTEGER`

The order of the matrix A. `N` >= 0.

AP Input and output parameter.

`AP` is `COMPLEX`

`AP` is an array, dimension (N*(N+1)/2). On entry, the triangular factor U or L from the Cholesky factorization $A = U^H * U$ or $A = L * L^H$, packed columnwise as a linear array. The j-th column of U or L is stored in the array `AP` as follows: if `UPLO` = 'U', AP(i + (j-1)*j/2) = U(i,j) for 1<=i<=j; if `UPLO` = 'L', AP(i + (j-1)*(2n-j)/2) = L(i,j) for j<=i<=n.

On exit, the upper or lower triangle of the (Hermitian) inverse of A, overwriting the input factor U or L.

INFO Output parameter.

`INFO` is `INTEGER`

= 0: successful exit < 0: if `INFO` = -i, the i-th argument had an illegal value > 0: if `INFO` = i, the (i,i) element of the factor U or L is zero, and the inverse could not be computed.

Related Information

For this routine in other precisions, please see *dpptri*, *spptri* and *zpptri*. It also exists with a native C interface as *LAPACKE_cpptri*.

4.4.11 csptri

csptri computes the inverse of a complex symmetric indefinite matrix A in packed storage using the factorization A = U*D*U^T or A = L*D*L^T computed by CSPTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine csptri(UPLO, N, AP, IPIV, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void csptri_(const char *uplo, const armpl_int_t *n,
             armpl_singlecomplex_t *ap, const armpl_int_t *ipiv,
             armpl_singlecomplex_t *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). On entry, the block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CSPTRF, stored as a packed triangular matrix.

On exit, if INFO = 0, the (symmetric) inverse of the original matrix, stored as a packed triangular matrix. The j-th column of inv(A) is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = inv(A)(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = inv(A)(i,j) for j<=i<=n.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CSPTRF.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *dsptri*, *ssptri* and *zsptri*. It also exists with a native C interface as *LAPACKE_csptri*.

4.4.12 csytri

`csytri` computes the inverse of a complex symmetric indefinite matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by CSYTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine csytri(UPLO, N, A, LDA, IPIV, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void csytri_(const char *uplo, const armpl_int_t *n, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, const armpl_int_t *ipiv,
             armpl_singlecomplex_t *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CSYTRF.

On exit, if INFO = 0, the (symmetric) inverse of the original matrix. If UPLO = 'U', the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; if UPLO = 'L' the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CSYTRF.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *dsytri*, *ssytri* and *zsytri*. It also exists with a native C interface as *LAPACKE_csytri*.

4.4.13 csytri2

csytri2 computes the inverse of a COMPLEX symmetric indefinite matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by CSYTRF. csytri2 sets the LEADING DIMENSION of the workspace before calling ‘‘csytri2‘‘X that actually computes the inverse.

Syntax

Fortran specification:

```
use armpl_library

subroutine csytri2(UPLO, N, A, LDA, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void csytri2_(const char *uplo, const armpl_int_t *n,
              armpl_singlecomplex_t *a, const armpl_int_t *lda,
              const armpl_int_t *ipiv, armpl_singlecomplex_t *work,
              const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = ‘U’: Upper triangular, form is A = U*D*U^T ; = ‘L’: Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the NB diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CSYTRF.

On exit, if INFO = 0, the (symmetric) inverse of the original matrix. If UPLO = 'U', the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; if UPLO = 'L' the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the NB structure of D as determined by CSYTRF.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (N+NB+1)*(NB+3) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. WORK is size >= (N+NB+1)*(NB+3) If LDWORK = -1, then a workspace query is assumed; the routine calculates: - the optimal size of the WORK array, returns this value as the first entry of the WORK array, - and no error message related to LDWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *dsytri2*, *ssytri2* and *zsytri2*. It also exists with a native C interface as *LAPACKE_csytri2*.

4.4.14 csytri2x

CSYTRI2X computes the inverse of a real symmetric indefinite matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by CSYTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine csytri2x(UPLO, N, A, LDA, IPIV, WORK, NB, INFO)
```

C specification:

```
#include "armpl.h"

void csytri2x_(const char *uplo, const armpl_int_t *n,
               armpl_singlecomplex_t *a, const armpl_int_t *lda,
               const armpl_int_t *ipiv, armpl_singlecomplex_t *work,
               const armpl_int_t *nb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the NNB diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CSYTRF.

On exit, if INFO = 0, the (symmetric) inverse of the original matrix. If UPLO = 'U', the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; if UPLO = 'L' the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the NNB structure of D as determined by CSYTRF.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (N+NB+1,NB+3) .

NB Input parameter.

NB is INTEGER

Block size

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *dsytri2x*, *ssytri2x* and *zsytri2x*. It also exists with a native C interface as *LAPACKE_csytri2x*.

4.4.15 csytri_3

CSYTRI_3 computes the inverse of a complex symmetric indefinite matrix A using the factorization computed by CSYTRF_RK or CSYTRF_BK:

```
A = P*U*D*(U**T)*(P**T) or A = P*L*D*(L**T)*(P**T),
```

where U (or L) is unit upper (or lower) triangular matrix, U^T (or L^T) is the transpose of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

CSYTRI_3 sets the leading dimension of the workspace before calling CSYTRI_3X that actually computes the inverse. This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine csytri_3(UPLO, N, A, LDA, E, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void csytri_3_(const char *uplo, const armpl_int_t *n,
               armpl_singlecomplex_t *a, const armpl_int_t *lda,
               const armpl_singlecomplex_t *e, const armpl_int_t *ipiv,
               armpl_singlecomplex_t *work, const armpl_int_t *lwork,
               armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, diagonal of the block diagonal matrix D and factors U or L as computed by CSYTRF_RK and CSYTRF_BK: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D should be provided on entry in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

On exit, if INFO = 0, the symmetric inverse of the original matrix. If UPLO = 'U': the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; If UPLO = 'L': the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Input parameter.

E is COMPLEX

E is an array, dimension (N). On entry, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i),i=2:N, E(1) not referenced; If UPLO = 'L': E(i) = D(i+1,i),i=1:N-1, E(N) not referenced.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is not referenced in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CSYTRF_RK or CSYTRF_BK.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (N+NB+1)*(NB+3).. On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= (N+NB+1)*(NB+3).

If LDWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *dsytri_3*, *ssytri_3* and *zsytri_3*. It also exists with a native C interface as *LAPACKE_csytri_3*.

4.4.16 csytri_rook

CSYTRI_ROOK computes the inverse of a complex symmetric matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by CSYTRF_ROOK.

Syntax

Fortran specification:

```
use armpl_library

subroutine csytri_rook(UPLO, N, A, LDA, IPIV, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void csytri_rook_(const char *uplo, const armpl_int_t *n,
                  armpl_singlecomplex_t *a, const armpl_int_t *lda,
                  const armpl_int_t *ipiv, armpl_singlecomplex_t *work,
                  armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CSYTRF_ROOK.

On exit, if INFO = 0, the (symmetric) inverse of the original matrix. If UPLO = 'U', the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; if UPLO = 'L' the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CSYTRF_ROOK.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *dsytri_rook*, *ssytri_rook* and *zsytri_rook*.

4.4.17 ctftri

ctftri computes the inverse of a triangular matrix A stored in RFP format.

This is a Level 3 BLAS version of the algorithm.

Syntax

Fortran specification:

```
use armpl_library

subroutine ctftri(TRANSR, UPLO, DIAG, N, A, INFO)
```

C specification:

```
#include "armpl.h"

void ctftri_(const char *transr, const char *uplo, const char *diag,
             const armpl_int_t *n, armpl_singlecomplex_t *a,
             armpl_int_t *info, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': The Normal TRANSR of RFP A is stored; = 'C': The Conjugate-transpose TRANSR of RFP A is stored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (N*(N+1)/2);. On entry, the triangular matrix A in RFP format. RFP format is described by TRANSR, UPLO, and N as follows: If TRANSR = 'N' then RFP A is (0:N,0:k-1) when N is even; k=N/2. RFP A is (0:N-1,0:k) when N is odd; k=N/2. IF TRANSR = 'C' then RFP is the Conjugate-transpose of RFP A as defined when TRANSR = 'N'. The contents of RFP A are defined by UPLO as follows: If UPLO = 'U' the RFP A contains the nt elements of upper packed A; If UPLO = 'L' the RFP A contains the nt elements of lower packed A. The LDA of RFP A is (N+1)/2 when TRANSR = 'C'. When TRANSR is 'N' the LDA is N+1 when N is even and N is odd. See the Note below for more details.

On exit, the (triangular) inverse of the original matrix, in the same storage format.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, A(i,i) is exactly zero. The triangular matrix is singular and its inverse can not be computed.

Related Information

For this routine in other precisions, please see *dtftri*, *stftri* and *ztftri*. It also exists with a native C interface as *LAPACKE_ctftri*.

4.4.18 ctptri

`ctptri` computes the inverse of a complex upper or lower triangular matrix A stored in packed format.

Syntax

Fortran specification:

```
use armpl_library

subroutine ctptri(UPLO, DIAG, N, AP, INFO)
```

C specification:

```
#include "armpl.h"

void ctptri_(const char *uplo, const char *diag, const armpl_int_t *n,
             armpl_singlecomplex_t *ap, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

= 'U': A is upper triangular; = 'L': A is lower triangular.

DIAG Input parameter.

`DIAG` is `CHARACTER*1`

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

`N` is `INTEGER`

The order of the matrix A. `N` >= 0.

AP Input and output parameter.

`AP` is `COMPLEX`

`AP` is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangular matrix A, stored columnwise in a linear array. The j-th column of A is stored in the array `AP` as follows: if `UPLO` = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if `UPLO` = 'L', AP(i + (j-1)*((2*n-j)/2) = A(i,j) for j<=i<=n. See below for further details. On exit, the (triangular) inverse of the original matrix, in the same packed storage format.

INFO Output parameter.

`INFO` is `INTEGER`

= 0: successful exit < 0: if `INFO` = -i, the i-th argument had an illegal value > 0: if `INFO` = i, A(i,i) is exactly zero. The triangular matrix is singular and its inverse can not be computed.

Related Information

For this routine in other precisions, please see *dtptri*, *stptri* and *ztptri*. It also exists with a native C interface as *LAPACKE_ctptri*.

4.4.19 ctrtri

`ctrtri` computes the inverse of a complex upper or lower triangular matrix A.

This is the Level 3 BLAS version of the algorithm.

Syntax

Fortran specification:

```
use armpl_library

subroutine ctrtri(UPLO, DIAG, N, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void ctrtri_(const char *uplo, const char *diag, const armpl_int_t *n,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1. On exit, the (triangular) inverse of the original matrix, in the same storage format.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, A(i,i) is exactly zero. The triangular matrix is singular and its inverse can not be computed.

Related Information

For this routine in other precisions, please see *dtrtri*, *strtri* and *ztrtri*. It also exists with a native C interface as *LAPACKE_ctrtri*.

4.4.20 dgetri

`dgetri` computes the inverse of a matrix using the LU factorization computed by DGETRF.

This method inverts U and then computes inv(A) by solving the system inv(A)*L = inv(U) for inv(A).

Syntax

Fortran specification:

```
use armpl_library

subroutine dgetri(N, A, LDA, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgetri_(const armpl_int_t *n, double *a, const armpl_int_t *lda,
             const armpl_int_t *ipiv, double *work, const armpl_int_t *lwork,
             armpl_int_t *info);
```

Parameters

N Input parameter.

`N` is `INTEGER`

The order of the matrix `A`. `N` >= 0.

A Input and output parameter.

`A` is `DOUBLE PRECISION`

`A` is an array, dimension (`LDA`, `N`). On entry, the factors L and U from the factorization `A` = P*L*U as computed by DGETRF. On exit, if `INFO` = 0, the inverse of the original matrix `A`.

LDA Input parameter.

`LDA` is `INTEGER`

The leading dimension of the array `A`. `LDA` >= max(1, `N`).

IPIV Input parameter.

`IPIV` is `INTEGER` array, dimension (`N`)

The pivot indices from DGETRF; for 1<=i<=N, row i of the matrix was interchanged with row IPIV(i).

WORK Output parameter.

`WORK` is `DOUBLE PRECISION`

`WORK` is an array, dimension (MAX(1, `LWORK`)). On exit, if INFO=0, then WORK(1) returns the optimal `LWORK`.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N). For optimal performance LWORK >= N*NB, where NB is the optimal blocksize returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, U(i,i) is exactly zero; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *cgetri*, *sgetri* and *zgetri*. It also exists with a native C interface as *LAPACKE_dgetri*.

4.4.21 dpftri

dpftri computes the inverse of a (real) symmetric positive definite matrix A using the Cholesky factorization A = U^T *U or A = L*L^T computed by DPFTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine dpftri(TRANSR, UPLO, N, A, INFO)
```

C specification:

```
#include "armpl.h"

void dpftri_(const char *transr, const char *uplo, const armpl_int_t *n,
             double *a, armpl_int_t *info, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': The Normal TRANSR of RFP A is stored; = 'T': The Transpose TRANSR of RFP A is stored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (N*(N+1)/2). On entry, the symmetric matrix A in RFP format. RFP format is described by TRANSR, UPLO, and N as follows: If TRANSR = 'N' then RFP A is (0:N,0:k-1) when N is even; k=N/2. RFP A is (0:N-1,0:k) when N is odd; k=N/2. IF TRANSR = 'T' then RFP is the transpose of RFP A as defined when TRANSR = 'N'. The contents of RFP A are defined by UPLO as follows: If UPLO = 'U' the RFP A contains the nt elements of upper packed A. If UPLO = 'L' the RFP A contains the elements of lower packed A. The LDA of RFP A is (N+1)/2 when TRANSR = 'T'. When TRANSR is 'N' the LDA is N+1 when N is even and N is odd. See the Note below for more details.

On exit, the symmetric inverse of the original matrix, in the same storage format.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the (i,i) element of the factor U or L is zero, and the inverse could not be computed.

Related Information

For this routine in other precisions, please see *cpftri*, *spftri* and *zpftri*. It also exists with a native C interface as *LAPACKE_dpftri*.

4.4.22 dpotri

dpotri computes the inverse of a real symmetric positive definite matrix A using the Cholesky factorization A = U^T *U or A = L*L^T computed by DPOTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine dpotri(UPLO, N, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void dpotri_(const char *uplo, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , as computed by DPOTRF. On exit, the upper or lower triangle of the (symmetric) inverse of A, overwriting the input factor U or L.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the (i,i) element of the factor U or L is zero, and the inverse could not be computed.

Related Information

For this routine in other precisions, please see *cpotri*, *spotri* and *zpotri*. It also exists with a native C interface as *LAPACKE_dpotri*.

4.4.23 dpptri

dpptri computes the inverse of a real symmetric positive definite matrix A using the Cholesky factorization A = U^T *U or A = L*L^T computed by DPPTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine dpptri(UPLO, N, AP, INFO)
```

C specification:

```
#include "armpl.h"

void dpptri_(const char *uplo, const armpl_int_t *n, double *ap,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangular factor is stored in AP; = 'L': Lower triangular factor is stored in AP.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is DOUBLE PRECISION

AP is an array, dimension (N*(N+1)/2). On entry, the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , packed columnwise as a linear array. The j-th column of U or L is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = U(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = L(i,j) for j<=i<=n.

On exit, the upper or lower triangle of the (symmetric) inverse of A, overwriting the input factor U or L.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the (i,i) element of the factor U or L is zero, and the inverse could not be computed.

Related Information

For this routine in other precisions, please see *cpptri*, *spptri* and *zpptri*. It also exists with a native C interface as *LAPACKE_dpptri*.

4.4.24 dsptri

dsptri computes the inverse of a real symmetric indefinite matrix A in packed storage using the factorization A = U*D*U^T or A = L*D*L^T computed by DSPTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsptri(UPLO, N, AP, IPIV, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dsptri_(const char *uplo, const armpl_int_t *n, double *ap,
             const armpl_int_t *ipiv, double *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is DOUBLE PRECISION

AP is an array, dimension (N*(N+1)/2). On entry, the block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by DSPTRF, stored as a packed triangular matrix.

On exit, if INFO = 0, the (symmetric) inverse of the original matrix, stored as a packed triangular matrix. The j-th column of inv(A) is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = inv(A)(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = inv(A)(i,j) for j<=i<=n.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by DSPTRF.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *csptri*, *ssptri* and *zsptri*. It also exists with a native C interface as *LAPACKE_dsptri*.

4.4.25 dsytri

dsytri computes the inverse of a real symmetric indefinite matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by DSYTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsytri(UPLO, N, A, LDA, IPIV, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dsytri_(const char *uplo, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, const armpl_int_t *ipiv, double *work,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by DSYTRF.

On exit, if INFO = 0, the (symmetric) inverse of the original matrix. If UPLO = 'U', the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; if UPLO = 'L' the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by DSYTRF.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *csytri*, *ssytri* and *zsytri*. It also exists with a native C interface as *LAPACKE_dsytri*.

4.4.26 dsytri2

dsytri2 computes the inverse of a DOUBLE PRECISION symmetric indefinite matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by DSYTRF. dsytri2 sets the LEADING DIMENSION of the workspace before calling ‘‘dsytri2‘‘X that actually computes the inverse.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsytri2(UPLO, N, A, LDA, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dsytri2_(const char *uplo, const armpl_int_t *n, double *a,
              const armpl_int_t *lda, const armpl_int_t *ipiv, double *work,
              const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the NB diagonal matrix D and the multipliers used to obtain the factor U or L as computed by DSYTRF.

On exit, if INFO = 0, the (symmetric) inverse of the original matrix. If UPLO = 'U', the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; if UPLO = 'L' the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the NB structure of D as determined by DSYTRF.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (N+NB+1)*(NB+3) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. WORK is size >= (N+NB+1)*(NB+3) If LDWORK = -1, then a workspace query is assumed; the routine calculates: - the optimal size of the WORK array, returns this value as the first entry of the WORK array, - and no error message related to LDWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *csytri2*, *ssytri2* and *zsytri2*. It also exists with a native C interface as *LAPACKE_dsytri2*.

4.4.27 dsytri2x

DSYTRI2X computes the inverse of a real symmetric indefinite matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by DSYTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsytri2x(UPLO, N, A, LDA, IPIV, WORK, NB, INFO)
```

C specification:

```
#include "armpl.h"

void dsytri2x_(const char *uplo, const armpl_int_t *n, double *a,
               const armpl_int_t *lda, const armpl_int_t *ipiv, double *work,
               const armpl_int_t *nb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the NNB diagonal matrix D and the multipliers used to obtain the factor U or L as computed by DSYTRF.

On exit, if INFO = 0, the (symmetric) inverse of the original matrix. If UPLO = 'U', the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; if UPLO = 'L' the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the NNB structure of D as determined by DSYTRF.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (N+NB+1,NB+3) .

NB Input parameter.

NB is INTEGER

Block size

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *csytri2x*, *ssytri2x* and *zsytri2x*. It also exists with a native C interface as *LAPACKE_dsytri2x*.

4.4.28 dsytri_3

DSYTRI_3 computes the inverse of a real symmetric indefinite matrix A using the factorization computed by DSYTRF_RK or DSYTRF_BK:

```
A = P*U*D*(U**T)*(P**T) or A = P*L*D*(L**T)*(P**T),
```

where U (or L) is unit upper (or lower) triangular matrix, U^T (or L^T) is the transpose of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

DSYTRI_3 sets the leading dimension of the workspace before calling DSYTRI_3X that actually computes the inverse. This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsytri_3(UPLO, N, A, LDA, E, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dsytri_3_(const char *uplo, const armpl_int_t *n, double *a,
               const armpl_int_t *lda, const double *e,
               const armpl_int_t *ipiv, double *work,
               const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, diagonal of the block diagonal matrix D and factors U or L as computed by DSYTRF_RK and DSYTRF_BK: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D should be provided on entry in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

On exit, if INFO = 0, the symmetric inverse of the original matrix. If UPLO = 'U': the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; If UPLO = 'L': the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Input parameter.

E is DOUBLE PRECISION

E is an array, dimension (N). On entry, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i),i=2:N, E(1) not referenced; If UPLO = 'L': E(i) = D(i+1,i),i=1:N-1, E(N) not referenced.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is not referenced in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by DSYTRF_RK or DSYTRF_BK.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (N+NB+1)*(NB+3).. On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= (N+NB+1)*(NB+3).

If LDWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *csytri_3*, *ssytri_3* and *zsytri_3*. It also exists with a native C interface as *LAPACKE_dsytri_3*.

4.4.29 dsytri_rook

DSYTRI_ROOK computes the inverse of a real symmetric matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by DSYTRF_ROOK.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsytri_rook(UPLO, N, A, LDA, IPIV, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dsytri_rook_(const char *uplo, const armpl_int_t *n, double *a,
                  const armpl_int_t *lda, const armpl_int_t *ipiv,
                  double *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by DSYTRF_ROOK.

On exit, if INFO = 0, the (symmetric) inverse of the original matrix. If UPLO = 'U', the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; if UPLO = 'L' the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by DSYTRF_ROOK.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *csytri_rook*, *ssytri_rook* and *zsytri_rook*.

4.4.30 dtftri

dtftri computes the inverse of a triangular matrix A stored in RFP format.

This is a Level 3 BLAS version of the algorithm.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtftri(TRANSR, UPLO, DIAG, N, A, INFO)
```

C specification:

```
#include "armpl.h"

void dtftri_(const char *transr, const char *uplo, const char *diag,
             const armpl_int_t *n, double *a, armpl_int_t *info, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': The Normal TRANSR of RFP A is stored; = 'T': The Transpose TRANSR of RFP A is stored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (0:nt-1);. nt=N*(N+1)/2. On entry, the triangular factor of a Hermitian Positive Definite matrix A in RFP format. RFP format is described by TRANSR, UPLO, and N as follows: If TRANSR = 'N' then RFP A is (0:N,0:k-1) when N is even; k=N/2. RFP A is (0:N-1,0:k) when N is odd; k=N/2. IF TRANSR = 'T' then RFP is the transpose of RFP A as defined when TRANSR = 'N'. The contents of RFP A are defined by UPLO as follows: If UPLO = 'U' the RFP A contains the nt elements of upper packed A; If UPLO = 'L' the RFP A contains the nt elements of lower packed A. The LDA of RFP A is (N+1)/2 when TRANSR = 'T'. When TRANSR is 'N' the LDA is N+1 when N is even and N is odd. See the Note below for more details.

On exit, the (triangular) inverse of the original matrix, in the same storage format.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, A(i,i) is exactly zero. The triangular matrix is singular and its inverse can not be computed.

Related Information

For this routine in other precisions, please see *ctftri*, *stftri* and *ztftri*. It also exists with a native C interface as *LAPACKE_dtftri*.

4.4.31 dtptri

dtptri computes the inverse of a real upper or lower triangular matrix A stored in packed format.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtptri(UPLO, DIAG, N, AP, INFO)
```

C specification:

```
#include "armpl.h"

void dtptri_(const char *uplo, const char *diag, const armpl_int_t *n,
             double *ap, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is DOUBLE PRECISION

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangular matrix A, stored columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*((2*n-j)/2) = A(i,j) for j<=i<=n. See below for further details. On exit, the (triangular) inverse of the original matrix, in the same packed storage format.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, A(i,i) is exactly zero. The triangular matrix is singular and its inverse can not be computed.

Related Information

For this routine in other precisions, please see *ctptri*, *stptri* and *ztptri*. It also exists with a native C interface as *LAPACKE_dtptri*.

4.4.32 dtrtri

dtrtri computes the inverse of a real upper or lower triangular matrix A.

This is the Level 3 BLAS version of the algorithm.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtrtri(UPLO, DIAG, N, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void dtrtri_(const char *uplo, const char *diag, const armpl_int_t *n,
             double *a, const armpl_int_t *lda, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1. On exit, the (triangular) inverse of the original matrix, in the same storage format.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, A(i,i) is exactly zero. The triangular matrix is singular and its inverse can not be computed.

Related Information

For this routine in other precisions, please see *ctrtri*, *strtri* and *ztrtri*. It also exists with a native C interface as *LAPACKE_dtrtri*.

4.4.33 sgetri

`sgetri` computes the inverse of a matrix using the LU factorization computed by SGETRF.

This method inverts U and then computes inv(A) by solving the system inv(A)*L = inv(U) for inv(A).

Syntax

Fortran specification:

```
use armpl_library

subroutine sgetri(N, A, LDA, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgetri_(const armpl_int_t *n, float *a, const armpl_int_t *lda,
             const armpl_int_t *ipiv, float *work, const armpl_int_t *lwork,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the factors L and U from the factorization A = P*L*U as computed by SGETRF. On exit, if INFO = 0, the inverse of the original matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from SGETRF; for 1<=i<=N, row i of the matrix was interchanged with row IPIV(i).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO=0, then WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N). For optimal performance LWORK >= N*NB, where NB is the optimal blocksize returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, U(i,i) is exactly zero; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *cgetri*, *dgetri* and *zgetri*. It also exists with a native C interface as *LAPACKE_sgetri*.

4.4.34 spftri

spftri computes the inverse of a real (symmetric) positive definite matrix A using the Cholesky factorization A = U^T *U or A = L*L^T computed by SPFTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine spftri(TRANSR, UPLO, N, A, INFO)
```

C specification:

```
#include "armpl.h"

void spftri_(const char *transr, const char *uplo, const armpl_int_t *n,
             float *a, armpl_int_t *info, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': The Normal TRANSR of RFP A is stored; = 'T': The Transpose TRANSR of RFP A is stored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (N*(N+1)/2). On entry, the symmetric matrix A in RFP format. RFP format is described by TRANSR, UPLO, and N as follows: If TRANSR = 'N' then RFP A is (0:N,0:k-1) when N is even; k=N/2. RFP A is (0:N-1,0:k) when N is odd; k=N/2. IF TRANSR = 'T' then RFP is the transpose of RFP A as defined when TRANSR = 'N'. The contents of RFP A are defined by UPLO as follows: If UPLO = 'U' the RFP A contains the nt elements of upper packed A. If UPLO = 'L' the RFP A contains the elements of lower packed A. The LDA of RFP A is (N+1)/2 when TRANSR = 'T'. When TRANSR is 'N' the LDA is N+1 when N is even and N is odd. See the Note below for more details.

On exit, the symmetric inverse of the original matrix, in the same storage format.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the (i,i) element of the factor U or L is zero, and the inverse could not be computed.

Related Information

For this routine in other precisions, please see *cpftri*, *dpftri* and *zpftri*. It also exists with a native C interface as *LAPACKE_spftri*.

4.4.35 spotri

spotri computes the inverse of a real symmetric positive definite matrix A using the Cholesky factorization A = U^T *U or A = L*L^T computed by SPOTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine spotri(UPLO, N, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void spotri_(const char *uplo, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , as computed by SPOTRF. On exit, the upper or lower triangle of the (symmetric) inverse of A, overwriting the input factor U or L.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the (i,i) element of the factor U or L is zero, and the inverse could not be computed.

Related Information

For this routine in other precisions, please see *cpotri*, *dpotri* and *zpotri*. It also exists with a native C interface as *LAPACKE_spotri*.

4.4.36 spptri

spptri computes the inverse of a real symmetric positive definite matrix A using the Cholesky factorization A = U^T *U or A = L*L^T computed by SPPTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine spptri(UPLO, N, AP, INFO)
```

C specification:

```
#include "armpl.h"

void spptri_(const char *uplo, const armpl_int_t *n, float *ap,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangular factor is stored in AP; = 'L': Lower triangular factor is stored in AP.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is REAL

AP is an array, dimension (N*(N+1)/2). On entry, the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , packed columnwise as a linear array. The j-th column of U or L is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = U(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = L(i,j) for j<=i<=n.

On exit, the upper or lower triangle of the (symmetric) inverse of A, overwriting the input factor U or L.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the (i,i) element of the factor U or L is zero, and the inverse could not be computed.

Related Information

For this routine in other precisions, please see *cpptri*, *dpptri* and *zpptri*. It also exists with a native C interface as *LAPACKE_spptri*.

4.4.37 ssptri

ssptri computes the inverse of a real symmetric indefinite matrix A in packed storage using the factorization A = U*D*U^T or A = L*D*L^T computed by SSPTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssptri(UPLO, N, AP, IPIV, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void ssptri_(const char *uplo, const armpl_int_t *n, float *ap,
             const armpl_int_t *ipiv, float *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is REAL

AP is an array, dimension (N*(N+1)/2). On entry, the block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by SSPTRF, stored as a packed triangular matrix.

On exit, if INFO = 0, the (symmetric) inverse of the original matrix, stored as a packed triangular matrix. The j-th column of inv(A) is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = inv(A)(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = inv(A)(i,j) for j<=i<=n.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by SSPTRF.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *csptri*, *dsptri* and *zsptri*. It also exists with a native C interface as *LAPACKE_ssptri*.

4.4.38 ssytri

ssytri computes the inverse of a real symmetric indefinite matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by SSYTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssytri(UPLO, N, A, LDA, IPIV, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void ssytri_(const char *uplo, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, const armpl_int_t *ipiv, float *work,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by SSYTRF.

On exit, if INFO = 0, the (symmetric) inverse of the original matrix. If UPLO = 'U', the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; if UPLO = 'L' the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by SSYTRF.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *csytri*, *dsytri* and *zsytri*. It also exists with a native C interface as *LAPACKE_ssytri*.

4.4.39 ssytri2

`ssytri2` computes the inverse of a REAL symmetric indefinite matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by SSYTRF. `ssytri2` sets the LEADING DIMENSION of the workspace before calling ‘‘ssytri2‘‘X that actually computes the inverse.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssytri2(UPLO, N, A, LDA, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ssytri2_(const char *uplo, const armpl_int_t *n, float *a,
              const armpl_int_t *lda, const armpl_int_t *ipiv, float *work,
              const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = ‘U’: Upper triangular, form is A = U*D*U^T ; = ‘L’: Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the NB diagonal matrix D and the multipliers used to obtain the factor U or L as computed by SSYTRF.

On exit, if INFO = 0, the (symmetric) inverse of the original matrix. If UPLO = ‘U’, the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; if UPLO = ‘L’ the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the NB structure of D as determined by SSYTRF.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (N+NB+1)*(NB+3) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. WORK is size >= (N+NB+1)*(NB+3) If LDWORK = -1, then a workspace query is assumed; the routine calculates: - the optimal size of the WORK array, returns this value as the first entry of the WORK array, - and no error message related to LDWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *csytri2*, *dsytri2* and *zsytri2*. It also exists with a native C interface as *LAPACKE_ssytri2*.

4.4.40 ssytri2x

SSYTRI2X computes the inverse of a real symmetric indefinite matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by SSYTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssytri2x(UPLO, N, A, LDA, IPIV, WORK, NB, INFO)
```

C specification:

```
#include "armpl.h"

void ssytri2x_(const char *uplo, const armpl_int_t *n, float *a,
               const armpl_int_t *lda, const armpl_int_t *ipiv, float *work,
               const armpl_int_t *nb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the NNB diagonal matrix D and the multipliers used to obtain the factor U or L as computed by SSYTRF.

On exit, if INFO = 0, the (symmetric) inverse of the original matrix. If UPLO = 'U', the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; if UPLO = 'L' the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the NNB structure of D as determined by SSYTRF.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (N+NB+1,NB+3) .

NB Input parameter.

NB is INTEGER

Block size

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *csytri2x*, *dsytri2x* and *zsytri2x*. It also exists with a native C interface as *LAPACKE_ssytri2x*.

4.4.41 ssytri_3

SSYTRI_3 computes the inverse of a real symmetric indefinite matrix A using the factorization computed by SSYTRF_RK or SSYTRF_BK:

```
A = P*U*D*(U**T)*(P**T) or A = P*L*D*(L**T)*(P**T),
```

where U (or L) is unit upper (or lower) triangular matrix, U^T (or L^T) is the transpose of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

SSYTRI_3 sets the leading dimension of the workspace before calling SSYTRI_3X that actually computes the inverse. This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssytri_3(UPLO, N, A, LDA, E, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ssytri_3_(const char *uplo, const armpl_int_t *n, float *a,
               const armpl_int_t *lda, const float *e,
               const armpl_int_t *ipiv, float *work, const armpl_int_t *lwork,
               armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, diagonal of the block diagonal matrix D and factors U or L as computed by SSYTRF_RK and SSYTRF_BK: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D should be provided on entry in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

On exit, if INFO = 0, the symmetric inverse of the original matrix. If UPLO = 'U': the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; If UPLO = 'L': the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Input parameter.

E is REAL

E is an array, dimension (N). On entry, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i),i=2:N, E(1) not referenced; If UPLO = 'L': E(i) = D(i+1,i),i=1:N-1, E(N) not referenced.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is not referenced in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by SSYTRF_RK or SSYTRF_BK.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (N+NB+1)*(NB+3).. On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= (N+NB+1)*(NB+3).

If LDWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *csytri_3*, *dsytri_3* and *zsytri_3*. It also exists with a native C interface as *LAPACKE_ssytri_3*.

4.4.42 ssytri_rook

SSYTRI_ROOK computes the inverse of a real symmetric matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by SSYTRF_ROOK.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssytri_rook(UPLO, N, A, LDA, IPIV, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void ssytri_rook_(const char *uplo, const armpl_int_t *n, float *a,
                  const armpl_int_t *lda, const armpl_int_t *ipiv,
                  float *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by SSYTRF_ROOK.

On exit, if INFO = 0, the (symmetric) inverse of the original matrix. If UPLO = 'U', the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; if UPLO = 'L' the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by SSYTRF_ROOK.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *csytri_rook*, *dsytri_rook* and *zsytri_rook*.

4.4.43 stftri

stftri computes the inverse of a triangular matrix A stored in RFP format.

This is a Level 3 BLAS version of the algorithm.

Syntax

Fortran specification:

```
use armpl_library

subroutine stftri(TRANSR, UPLO, DIAG, N, A, INFO)
```

C specification:

```
#include "armpl.h"

void stftri_(const char *transr, const char *uplo, const char *diag,
             const armpl_int_t *n, float *a, armpl_int_t *info, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': The Normal TRANSR of RFP A is stored; = 'T': The Transpose TRANSR of RFP A is stored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (NT);. NT=N*(N+1)/2. On entry, the triangular factor of a Hermitian Positive Definite matrix A in RFP format. RFP format is described by TRANSR, UPLO, and N as follows: If TRANSR = 'N' then RFP A is (0:N,0:k-1) when N is even; k=N/2. RFP A is (0:N-1,0:k) when N is odd; k=N/2. IF TRANSR = 'T' then RFP is the transpose of RFP A as defined when TRANSR = 'N'. The contents of RFP A are defined by UPLO as follows: If UPLO = 'U' the RFP A contains the nt elements of upper packed A; If UPLO = 'L' the RFP A contains the nt elements of lower packed A. The LDA of RFP A is (N+1)/2 when TRANSR = 'T'. When TRANSR is 'N' the LDA is N+1 when N is even and N is odd. See the Note below for more details.

On exit, the (triangular) inverse of the original matrix, in the same storage format.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, A(i,i) is exactly zero. The triangular matrix is singular and its inverse can not be computed.

Related Information

For this routine in other precisions, please see *ctftri*, *dtftri* and *ztftri*. It also exists with a native C interface as *LAPACKE_stftri*.

4.4.44 stptri

stptri computes the inverse of a real upper or lower triangular matrix A stored in packed format.

Syntax

Fortran specification:

```
use armpl_library

subroutine stptri(UPLO, DIAG, N, AP, INFO)
```

C specification:

```
#include "armpl.h"

void stptri_(const char *uplo, const char *diag, const armpl_int_t *n,
             float *ap, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is REAL

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangular matrix A, stored columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*((2*n-j)/2) = A(i,j) for j<=i<=n. See below for further details. On exit, the (triangular) inverse of the original matrix, in the same packed storage format.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, A(i,i) is exactly zero. The triangular matrix is singular and its inverse can not be computed.

Related Information

For this routine in other precisions, please see *ctptri*, *dtptri* and *ztptri*. It also exists with a native C interface as *LAPACKE_stptri*.

4.4.45 strtri

strtri computes the inverse of a real upper or lower triangular matrix A.

This is the Level 3 BLAS version of the algorithm.

Syntax

Fortran specification:

```
use armpl_library

subroutine strtri(UPLO, DIAG, N, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void strtri_(const char *uplo, const char *diag, const armpl_int_t *n,
             float *a, const armpl_int_t *lda, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1. On exit, the (triangular) inverse of the original matrix, in the same storage format.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, A(i,i) is exactly zero. The triangular matrix is singular and its inverse can not be computed.

Related Information

For this routine in other precisions, please see *ctrtri*, *dtrtri* and *ztrtri*. It also exists with a native C interface as *LAPACKE_strtri*.

4.4.46 zgetri

zgetri computes the inverse of a matrix using the LU factorization computed by ZGETRF.

This method inverts U and then computes inv(A) by solving the system inv(A)*L = inv(U) for inv(A).

Syntax

Fortran specification:

```
use armpl_library

subroutine zgetri(N, A, LDA, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgetri_(const armpl_int_t *n, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, const armpl_int_t *ipiv,
             armpl_doublecomplex_t *work, const armpl_int_t *lwork,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the factors L and U from the factorization A = P*L*U as computed by ZGETRF. On exit, if INFO = 0, the inverse of the original matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from ZGETRF; for 1<=i<=N, row i of the matrix was interchanged with row IPIV(i).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO=0, then WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N). For optimal performance LWORK >= N*NB, where NB is the optimal blocksize returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, U(i,i) is exactly zero; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *cgetri*, *dgetri* and *sgetri*. It also exists with a native C interface as *LAPACKE_zgetri*.

4.4.47 zhetri

`zhetri` computes the inverse of a complex Hermitian indefinite matrix A using the factorization A = U*D*U^H or A = L*D*L^H computed by ZHETRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhetri(UPLO, N, A, LDA, IPIV, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhetri_(const char *uplo, const armpl_int_t *n, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, const armpl_int_t *ipiv,
             armpl_doublecomplex_t *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^H ; = 'L': Lower triangular, form is A = L*D*L^H .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by ZHETRF.

On exit, if INFO = 0, the (Hermitian) inverse of the original matrix. If UPLO = 'U', the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; if UPLO = 'L' the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZHETRF.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *chetri*. It also exists with a native C interface as *LAPACKE_zhetri*.

4.4.48 zhetri2

zhetri2 computes the inverse of a COMPLEX*16 hermitian indefinite matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by ZHETRF. zhetri2 set the LEADING DIMENSION of the workspace before calling ‘‘zhetri2‘‘X that actually computes the inverse.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhetri2(UPLO, N, A, LDA, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhetri2_(const char *uplo, const armpl_int_t *n,
              armpl_doublecomplex_t *a, const armpl_int_t *lda,
              const armpl_int_t *ipiv, armpl_doublecomplex_t *work,
              const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = ‘U’: Upper triangular, form is A = U*D*U^T ; = ‘L’: Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the NB diagonal matrix D and the multipliers used to obtain the factor U or L as computed by ZHETRF.

On exit, if INFO = 0, the (symmetric) inverse of the original matrix. If UPLO = 'U', the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; if UPLO = 'L' the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the NB structure of D as determined by ZHETRF.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N+NB+1)*(NB+3) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. WORK is size >= (N+NB+1)*(NB+3) If LWORK = -1, then a workspace query is assumed; the routine calculates: - the optimal size of the WORK array, returns this value as the first entry of the WORK array, - and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *chetri2*. It also exists with a native C interface as *LAPACKE_zhetri2*.

4.4.49 zhetri2x

ZHETRI2X computes the inverse of a COMPLEX*16 Hermitian indefinite matrix A using the factorization A = U*D*U^H or A = L*D*L^H computed by ZHETRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhetri2x(UPLO, N, A, LDA, IPIV, WORK, NB, INFO)
```

C specification:

```
#include "armpl.h"

void zhetri2x_(const char *uplo, const armpl_int_t *n,
               armpl_doublecomplex_t *a, const armpl_int_t *lda,
               const armpl_int_t *ipiv, armpl_doublecomplex_t *work,
               const armpl_int_t *nb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^H ; = 'L': Lower triangular, form is A = L*D*L^H .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the NNB diagonal matrix D and the multipliers used to obtain the factor U or L as computed by ZHETRF.

On exit, if INFO = 0, the (symmetric) inverse of the original matrix. If UPLO = 'U', the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; if UPLO = 'L' the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the NNB structure of D as determined by ZHETRF.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N+NB+1,NB+3) .

NB Input parameter.

NB is INTEGER

Block size

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *chetri2x*. It also exists with a native C interface as *LAPACKE_zhetri2x*.

4.4.50 zhetri_3

ZHETRI_3 computes the inverse of a complex Hermitian indefinite matrix A using the factorization computed by ZHETRF_RK or ZHETRF_BK:

```
A = P*U*D*(U**H)*(P**T) or A = P*L*D*(L**H)*(P**T),
```

where U (or L) is unit upper (or lower) triangular matrix, U^H (or L^H) is the conjugate of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is Hermitian and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

ZHETRI_3 sets the leading dimension of the workspace before calling ZHETRI_3X that actually computes the inverse. This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhetri_3(UPLO, N, A, LDA, E, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhetri_3_(const char *uplo, const armpl_int_t *n,
               armpl_doublecomplex_t *a, const armpl_int_t *lda,
               const armpl_doublecomplex_t *e, const armpl_int_t *ipiv,
               armpl_doublecomplex_t *work, const armpl_int_t *lwork,
               armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, diagonal of the block diagonal matrix D and factors U or L as computed by ZHETRF_RK and ZHETRF_BK: a) ONLY diagonal elements of the Hermitian block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D should be provided on entry in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

On exit, if INFO = 0, the Hermitian inverse of the original matrix. If UPLO = 'U': the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; If UPLO = 'L': the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Input parameter.

E is COMPLEX*16

E is an array, dimension (N). On entry, contains the superdiagonal (or subdiagonal) elements of the Hermitian block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i),i=2:N, E(1) not referenced; If UPLO = 'L': E(i) = D(i+1,i),i=1:N-1, E(N) not referenced.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is not referenced in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZHETRF_RK or ZHETRF_BK.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N+NB+1)*(NB+3).. On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= (N+NB+1)*(NB+3).

If LDWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *chetri_3*. It also exists with a native C interface as *LAPACKE_zhetri_3*.

4.4.51 zhetri_rook

ZHETRI_ROOK computes the inverse of a complex Hermitian indefinite matrix A using the factorization A = $U*D*U^H$ or A = $L*D*L^H$ computed by ZHETRF_ROOK.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhetri_rook(UPLO, N, A, LDA, IPIV, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhetri_rook_(const char *uplo, const armpl_int_t *n,
                  armpl_doublecomplex_t *a, const armpl_int_t *lda,
                  const armpl_int_t *ipiv, armpl_doublecomplex_t *work,
                  armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^H ; = 'L': Lower triangular, form is A = L*D*L^H .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by ZHETRF_ROOK.

On exit, if INFO = 0, the (Hermitian) inverse of the original matrix. If UPLO = 'U', the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; if UPLO = 'L' the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZHETRF_ROOK.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *chetri_rook*.

4.4.52 zhptri

zhptri computes the inverse of a complex Hermitian indefinite matrix A in packed storage using the factorization A = U*D*U^H or A = L*D*L^H computed by ZHPTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhptri(UPLO, N, AP, IPIV, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhptri_(const char *uplo, const armpl_int_t *n,
             armpl_doublecomplex_t *ap, const armpl_int_t *ipiv,
             armpl_doublecomplex_t *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^H ; = 'L': Lower triangular, form is A = L*D*L^H .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). On entry, the block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by ZHPTRF, stored as a packed triangular matrix.

On exit, if INFO = 0, the (Hermitian) inverse of the original matrix, stored as a packed triangular matrix. The j-th column of inv(A) is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = inv(A)(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = inv(A)(i,j) for j<=i<=n.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZHPTRF.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *chptri*. It also exists with a native C interface as *LAPACKE_zhptri*.

4.4.53 zpftri

zpftri computes the inverse of a complex Hermitian positive definite matrix A using the Cholesky factorization A = U^H *U or A = L*L^H computed by ZPFTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine zpftri(TRANSR, UPLO, N, A, INFO)
```

C specification:

```
#include "armpl.h"

void zpftri_(const char *transr, const char *uplo, const armpl_int_t *n,
             armpl_doublecomplex_t *a, armpl_int_t *info, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': The Normal TRANSR of RFP A is stored; = 'C': The Conjugate-transpose TRANSR of RFP A is stored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (N*(N+1)/2);. On entry, the Hermitian matrix A in RFP format. RFP format is described by TRANSR, UPLO, and N as follows: If TRANSR = 'N' then RFP A is (0:N,0:k-1) when N is even; k=N/2. RFP A is (0:N-1,0:k) when N is odd; k=N/2. IF TRANSR = 'C' then RFP is the Conjugate-transpose of RFP A as defined when TRANSR = 'N'. The contents of RFP A are defined by UPLO as follows: If UPLO = 'U' the RFP A contains the nt elements of upper packed A. If UPLO = 'L' the RFP A contains the elements of lower packed A. The LDA of RFP A is (N+1)/2 when TRANSR = 'C'. When TRANSR is 'N' the LDA is N+1 when N is even and N is odd. See the Note below for more details.

On exit, the Hermitian inverse of the original matrix, in the same storage format.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the (i,i) element of the factor U or L is zero, and the inverse could not be computed.

Related Information

For this routine in other precisions, please see *cpftri*, *dpftri* and *spftri*. It also exists with a native C interface as *LAPACKE_zpftri*.

4.4.54 zpotri

zpotri computes the inverse of a complex Hermitian positive definite matrix A using the Cholesky factorization A = U^H *U or A = L*L^H computed by ZPOTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine zpotri(UPLO, N, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void zpotri_(const char *uplo, const armpl_int_t *n, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H , as computed by ZPOTRF. On exit, the upper or lower triangle of the (Hermitian) inverse of A, overwriting the input factor U or L.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the (i,i) element of the factor U or L is zero, and the inverse could not be computed.

Related Information

For this routine in other precisions, please see *cpotri*, *dpotri* and *spotri*. It also exists with a native C interface as *LAPACKE_zpotri*.

4.4.55 zpptri

zpptri computes the inverse of a complex Hermitian positive definite matrix A using the Cholesky factorization $A = U^H * U$ or $A = L * L^H$ computed by ZPPTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine zpptri(UPLO, N, AP, INFO)
```

C specification:

```
#include "armpl.h"

void zpptri_(const char *uplo, const armpl_int_t *n,
             armpl_doublecomplex_t *ap, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangular factor is stored in AP; = 'L': Lower triangular factor is stored in AP.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). On entry, the triangular factor U or L from the Cholesky factorization $A = U^H * U$ or $A = L * L^H$, packed columnwise as a linear array. The j-th column of U or L is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = U(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = L(i,j) for j<=i<=n.

On exit, the upper or lower triangle of the (Hermitian) inverse of A, overwriting the input factor U or L.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the (i,i) element of the factor U or L is zero, and the inverse could not be computed.

Related Information

For this routine in other precisions, please see *cpptri*, *dpptri* and *spptri*. It also exists with a native C interface as *LAPACKE_zpptri*.

4.4.56 zsptri

zsptri computes the inverse of a complex symmetric indefinite matrix A in packed storage using the factorization $A = U * D * U^T$ or $A = L * D * L^T$ computed by ZSPTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsptri(UPLO, N, AP, IPIV, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zsptri_(const char *uplo, const armpl_int_t *n,
             armpl_doublecomplex_t *ap, const armpl_int_t *ipiv,
             armpl_doublecomplex_t *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). On entry, the block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by ZSPTRF, stored as a packed triangular matrix.

On exit, if INFO = 0, the (symmetric) inverse of the original matrix, stored as a packed triangular matrix. The j-th column of inv(A) is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = inv(A)(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = inv(A)(i,j) for j<=i<=n.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZSPTRF.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *csptri*, *dsptri* and *ssptri*. It also exists with a native C interface as *LAPACKE_zsptri*.

4.4.57 zsytri

`zsytri` computes the inverse of a complex symmetric indefinite matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by ZSYTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsytri(UPLO, N, A, LDA, IPIV, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zsytri_(const char *uplo, const armpl_int_t *n, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, const armpl_int_t *ipiv,
             armpl_doublecomplex_t *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by ZSYTRF.

On exit, if INFO = 0, the (symmetric) inverse of the original matrix. If UPLO = 'U', the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; if UPLO = 'L' the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZSYTRF.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *csytri*, *dsytri* and *ssytri*. It also exists with a native C interface as *LAPACKE_zsytri*.

4.4.58 zsytri2

zsytri2 computes the inverse of a COMPLEX*16 symmetric indefinite matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by ZSYTRF. zsytri2 sets the LEADING DIMENSION of the workspace before calling ‘‘zsytri2‘‘X that actually computes the inverse.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsytri2(UPLO, N, A, LDA, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zsytri2_(const char *uplo, const armpl_int_t *n,
              armpl_doublecomplex_t *a, const armpl_int_t *lda,
              const armpl_int_t *ipiv, armpl_doublecomplex_t *work,
              const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = ‘U’: Upper triangular, form is A = U*D*U^T ; = ‘L’: Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the NB diagonal matrix D and the multipliers used to obtain the factor U or L as computed by ZSYTRF.

On exit, if INFO = 0, the (symmetric) inverse of the original matrix. If UPLO = ‘U’, the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; if UPLO = ‘L’ the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the NB structure of D as determined by ZSYTRF.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N+NB+1)*(NB+3) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. WORK is size >= (N+NB+1)*(NB+3) If LDWORK = -1, then a workspace query is assumed; the routine calculates: - the optimal size of the WORK array, returns this value as the first entry of the WORK array, - and no error message related to LDWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *csytri2*, *dsytri2* and *ssytri2*. It also exists with a native C interface as *LAPACKE_zsytri2*.

4.4.59 zsytri2x

ZSYTRI2X computes the inverse of a complex symmetric indefinite matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by ZSYTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsytri2x(UPLO, N, A, LDA, IPIV, WORK, NB, INFO)
```

C specification:

```
#include "armpl.h"

void zsytri2x_(const char *uplo, const armpl_int_t *n,
               armpl_doublecomplex_t *a, const armpl_int_t *lda,
               const armpl_int_t *ipiv, armpl_doublecomplex_t *work,
               const armpl_int_t *nb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the NNB diagonal matrix D and the multipliers used to obtain the factor U or L as computed by ZSYTRF.

On exit, if INFO = 0, the (symmetric) inverse of the original matrix. If UPLO = 'U', the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; if UPLO = 'L' the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the NNB structure of D as determined by ZSYTRF.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N+NB+1,NB+3) .

NB Input parameter.

NB is INTEGER

Block size

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *csytri2x*, *dsytri2x* and *ssytri2x*. It also exists with a native C interface as *LAPACKE_zsytri2x*.

4.4.60 zsytri_3

ZSYTRI_3 computes the inverse of a complex symmetric indefinite matrix A using the factorization computed by ZSYTRF_RK or ZSYTRF_BK:

```
A = P*U*D*(U**T)*(P**T) or A = P*L*D*(L**T)*(P**T),
```

where U (or L) is unit upper (or lower) triangular matrix, U^T (or L^T) is the transpose of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

ZSYTRI_3 sets the leading dimension of the workspace before calling ZSYTRI_3X that actually computes the inverse. This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsytri_3(UPLO, N, A, LDA, E, IPIV, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zsytri_3_(const char *uplo, const armpl_int_t *n,
               armpl_doublecomplex_t *a, const armpl_int_t *lda,
               const armpl_doublecomplex_t *e, const armpl_int_t *ipiv,
               armpl_doublecomplex_t *work, const armpl_int_t *lwork,
               armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, diagonal of the block diagonal matrix D and factors U or L as computed by ZSYTRF_RK and ZSYTRF_BK: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D should be provided on entry in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

On exit, if INFO = 0, the symmetric inverse of the original matrix. If UPLO = 'U': the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; If UPLO = 'L': the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Input parameter.

E is COMPLEX*16

E is an array, dimension (N). On entry, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i),i=2:N, E(1) not referenced; If UPLO = 'L': E(i) = D(i+1,i),i=1:N-1, E(N) not referenced.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is not referenced in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZSYTRF_RK or ZSYTRF_BK.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N+NB+1)*(NB+3).. On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= (N+NB+1)*(NB+3).

If LDWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *csytri_3*, *dsytri_3* and *ssytri_3*. It also exists with a native C interface as *LAPACKE_zsytri_3*.

4.4.61 zsytri_rook

ZSYTRI_ROOK computes the inverse of a complex symmetric matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by ZSYTRF_ROOK.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsytri_rook(UPLO, N, A, LDA, IPIV, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zsytri_rook_(const char *uplo, const armpl_int_t *n,
                  armpl_doublecomplex_t *a, const armpl_int_t *lda,
                  const armpl_int_t *ipiv, armpl_doublecomplex_t *work,
                  armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by ZSYTRF_ROOK.

On exit, if INFO = 0, the (symmetric) inverse of the original matrix. If UPLO = 'U', the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; if UPLO = 'L' the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZSYTRF_ROOK.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *csytri_rook*, *dsytri_rook* and *ssytri_rook*.

4.4.62 ztftri

ztftri computes the inverse of a triangular matrix A stored in RFP format.

This is a Level 3 BLAS version of the algorithm.

Syntax

Fortran specification:

```
use armpl_library

subroutine ztftri(TRANSR, UPLO, DIAG, N, A, INFO)
```

C specification:

```
#include "armpl.h"

void ztftri_(const char *transr, const char *uplo, const char *diag,
             const armpl_int_t *n, armpl_doublecomplex_t *a,
             armpl_int_t *info, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': The Normal TRANSR of RFP A is stored; = 'C': The Conjugate-transpose TRANSR of RFP A is stored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (N*(N+1)/2);. On entry, the triangular matrix A in RFP format. RFP format is described by TRANSR, UPLO, and N as follows: If TRANSR = 'N' then RFP A is (0:N,0:k-1) when N is even; k=N/2. RFP A is (0:N-1,0:k) when N is odd; k=N/2. IF TRANSR = 'C' then RFP is the Conjugate-transpose of RFP A as defined when TRANSR = 'N'. The contents of RFP A are defined by UPLO as follows: If UPLO = 'U' the RFP A contains the nt elements of upper packed A; If UPLO = 'L' the RFP A contains the nt elements of lower packed A. The LDA of RFP A is (N+1)/2 when TRANSR = 'C'. When TRANSR is 'N' the LDA is N+1 when N is even and N is odd. See the Note below for more details.

On exit, the (triangular) inverse of the original matrix, in the same storage format.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, A(i,i) is exactly zero. The triangular matrix is singular and its inverse can not be computed.

Related Information

For this routine in other precisions, please see *ctftri*, *dtftri* and *stftri*. It also exists with a native C interface as *LAPACKE_ztftri*.

4.4.63 ztptri

`ztptri` computes the inverse of a complex upper or lower triangular matrix A stored in packed format.

Syntax

Fortran specification:

```
use armpl_library

subroutine ztptri(UPLO, DIAG, N, AP, INFO)
```

C specification:

```
#include "armpl.h"

void ztptri_(const char *uplo, const char *diag, const armpl_int_t *n,
             armpl_doublecomplex_t *ap, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

= 'U': A is upper triangular; = 'L': A is lower triangular.

DIAG Input parameter.

`DIAG` is `CHARACTER*1`

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

`N` is `INTEGER`

The order of the matrix A. `N` >= 0.

AP Input and output parameter.

`AP` is `COMPLEX*16`

`AP` is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangular matrix A, stored columnwise in a linear array. The j-th column of A is stored in the array `AP` as follows: if `UPLO` = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if `UPLO` = 'L', AP(i + (j-1)*((2*n-j)/2) = A(i,j) for j<=i<=n. See below for further details. On exit, the (triangular) inverse of the original matrix, in the same packed storage format.

INFO Output parameter.

`INFO` is `INTEGER`

= 0: successful exit < 0: if `INFO` = -i, the i-th argument had an illegal value > 0: if `INFO` = i, A(i,i) is exactly zero. The triangular matrix is singular and its inverse can not be computed.

Related Information

For this routine in other precisions, please see *ctptri*, *dtptri* and *stptri*. It also exists with a native C interface as *LAPACKE_ztptri*.

4.4.64 ztrtri

`ztrtri` computes the inverse of a complex upper or lower triangular matrix A.

This is the Level 3 BLAS version of the algorithm.

Syntax

Fortran specification:

```
use armpl_library

subroutine ztrtri(UPLO, DIAG, N, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void ztrtri_(const char *uplo, const char *diag, const armpl_int_t *n,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1. On exit, the (triangular) inverse of the original matrix, in the same storage format.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, A(i,i) is exactly zero. The triangular matrix is singular and its inverse can not be computed.

Related Information

For this routine in other precisions, please see *ctrtri*, *dtrtri* and *strtri*. It also exists with a native C interface as *LAPACKE_ztrtri*.

4.5 LAPACK least squares routines

4.5.1 cgels

`cgels` solves overdetermined or underdetermined complex linear systems involving an M-by-N matrix A, or its conjugate-transpose, using a QR or LQ factorization of A. It is assumed that A has full rank.

The following options are provided:

1. If TRANS = '`N`' and m >= n: find the least squares solution of

```
an overdetermined system, i.e., solve the least squares problem
            minimize || B - A*X ||.
```

2. If TRANS = '`N`' and m < n: find the minimum norm solution of

```
an underdetermined system A * X = B.
```

3. If TRANS = '`C`' and m >= n: find the minimum norm solution of

```
an underdetermined system A**H * X = B.
```

4. If TRANS = '`C`' and m < n: find the least squares solution of

```
an overdetermined system, i.e., solve the least squares problem
            minimize || B - A**H * X ||.
```

Several right hand side vectors b and solution vectors x can be handled in a single call; they are stored as the columns of the M-by-NRHS right hand side matrix B and the N-by-NRHS solution matrix X.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgels(TRANS, M, N, NRHS, A, LDA, B, LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgels_(const char *trans, const armpl_int_t *m, const armpl_int_t *n,
            const armpl_int_t *nrhs, armpl_singlecomplex_t *a,
            const armpl_int_t *lda, armpl_singlecomplex_t *b,
            const armpl_int_t *ldb, armpl_singlecomplex_t *work,
            const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

`TRANS` Input parameter.

`TRANS` is `CHARACTER*1`

= 'N': the linear system involves A; = 'C': the linear system involves A^H .

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. if M >= N, A is overwritten by details of its QR factorization as returned by CGEQRF; if M < N, A is overwritten by details of its LQ factorization as returned by CGELQF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the matrix B of right hand side vectors, stored columnwise; B is M-by-NRHS if TRANS = 'N', or N-by-NRHS if TRANS = 'C'. On exit, if INFO = 0, B is overwritten by the solution vectors, stored columnwise: if TRANS = 'N' and m >= n, rows 1 to n of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of the modulus of elements N+1 to M in that column; if TRANS = 'N' and m < n, rows 1 to N of B contain the minimum norm solution vectors; if TRANS = 'C' and m >= n, rows 1 to M of B contain the minimum norm solution vectors; if TRANS = 'C' and m < n, rows 1 to M of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of the modulus of elements M+1 to N in that column.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= MAX(1, M, N).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, MN + max(MN, NRHS)). For optimal performance, LWORK >= max(1, MN + max(MN, NRHS)*NB). where MN = min(M, N) and NB is the optimum block size.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element of the triangular factor of A is zero, so that A does not have full rank; the least squares solution could not be computed.

Related Information

For this routine in other precisions, please see *dgels*, *sgels* and *zgels*. It also exists with a native C interface as *LAPACKE_cgels*.

4.5.2 cgelsd

cgelsd computes the minimum-norm solution to a real linear least squares problem:

```
minimize 2-norm(| b - A*x |)
```

using the singular value decomposition (SVD) of A. A is an M-by-N matrix which may be rank-deficient.

Several right hand side vectors b and solution vectors x can be handled in a single call; they are stored as the columns of the M-by-NRHS right hand side matrix B and the N-by-NRHS solution matrix X.

The problem is solved in three steps: (1) Reduce the coefficient matrix A to bidiagonal form with

```
Householder transformations, reducing the original problem
into a "bidiagonal least squares problem" (BLS)
```

(2) Solve the BLS using a divide and conquer approach.

(3) Apply back all the Householder transformations to solve

```
the original least squares problem.
```

The effective rank of A is determined by treating as zero those singular values which are less than RCOND times the largest singular value.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgelsd(M, N, NRHS, A, LDA, B, LDB, S, RCOND, RANK, WORK, LWORK,
                  RWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgelsd_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *nrhs, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, float *s, const float *rcond,
```

(continues on next page)

(continued from previous page)

```
            armpl_int_t *rank, armpl_singlecomplex_t *work,
            const armpl_int_t *lwork, float *rwork, armpl_int_t *iwork,
            armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, A has been destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the M-by-NRHS right hand side matrix B. On exit, B is overwritten by the N-by-NRHS solution matrix X. If m >= n and RANK = n, the residual sum-of-squares for the solution in the i-th column is given by the sum of squares of the modulus of elements n+1:m in that column.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M, N).

S Output parameter.

S is REAL

S is an array, dimension (min(M, N)). The singular values of A in decreasing order. The condition number of A in the 2-norm = S(1)/S(min(m,n)).

RCOND Input parameter.

RCOND is REAL

RCOND is used to determine the effective rank of A. Singular values S(i) <= RCOND*S(1) are treated as zero. If RCOND < 0, machine precision is used instead.

RANK Output parameter.

RANK is INTEGER

The effective rank of A, i.e., the number of singular values which are greater than RCOND*S(1).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK must be at least 1. The exact minimum amount of workspace needed depends on M, N and NRHS. As long as LWORK is at least 2 * N + N * NRHS if M is greater than or equal to N or 2 * M + M * NRHS if M is less than N, the code will execute correctly. For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the array WORK and the minimum sizes of the arrays RWORK and IWORK, and returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (MAX(1,LRWORK)). LRWORK >= 10* N + 2*N*SMLSIZ + 8*N*NLVL + 3*SMLSIZ* NRHS + MAX((SMLSIZ+1)**2, N*(1+NRHS) + 2* NRHS) if M is greater than or equal to N or 10* M + 2*M*SMLSIZ + 8*M*NLVL + 3*SMLSIZ* NRHS + MAX((SMLSIZ+1)**2, N*(1+NRHS) + 2* NRHS) if M is less than N, the code will execute correctly. SMLSIZ is returned by ILAENV and is equal to the maximum size of the subproblems at the bottom of the computation tree (usually about 25), and NLVL = MAX(0, INT(LOG_2(MIN(M,N)/(SMLSIZ+1))) + 1) On exit, if INFO = 0, RWORK(1) returns the minimum LRWORK.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1,LIWORK))

LIWORK >= max(1, 3*MINMN*NLVL + 11*MINMN), where MINMN = MIN(M,N). On exit, if INFO = 0, IWORK(1) returns the minimum LIWORK.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: the algorithm for computing the SVD failed to converge; if INFO = i, i off-diagonal elements of an intermediate bidiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *dgelsd*, *sgelsd* and *zgelsd*. It also exists with a native C interface as *LAPACKE_cgelsd*.

4.5.3 cgelss

cgelss computes the minimum norm solution to a complex linear least squares problem:

Minimize 2-norm(| b - A*x |).

using the singular value decomposition (SVD) of A. A is an M-by-N matrix which may be rank-deficient.

Several right hand side vectors b and solution vectors x can be handled in a single call; they are stored as the columns of the M-by-NRHS right hand side matrix B and the N-by-NRHS solution matrix X.

The effective rank of A is determined by treating as zero those singular values which are less than RCOND times the largest singular value.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgelss(M, N, NRHS, A, LDA, B, LDB, S, RCOND, RANK, WORK, LWORK,
                  RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgelss_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *nrhs, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, float *s, const float *rcond,
             armpl_int_t *rank, armpl_singlecomplex_t *work,
             const armpl_int_t *lwork, float *rwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the first min(m,n) rows of A are overwritten with its right singular vectors, stored rowwise.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the M-by-NRHS right hand side matrix B. On exit, B is overwritten by the N-by-NRHS solution matrix X. If m >= n and RANK = n, the residual sum-of-squares for the solution in the i-th column is given by the sum of squares of the modulus of elements n+1:m in that column.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M, N).

S Output parameter.

S is REAL

S is an array, dimension (min(M, N)). The singular values of A in decreasing order. The condition number of A in the 2-norm = S(1)/S(min(m,n)).

RCOND Input parameter.

RCOND is REAL

RCOND is used to determine the effective rank of A. Singular values S(i) <= RCOND*S(1) are treated as zero. If RCOND < 0, machine precision is used instead.

RANK Output parameter.

RANK is INTEGER

The effective rank of A, i.e., the number of singular values which are greater than RCOND*S(1).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= 1, and also: LWORK >= 2*min(M, N) + max(M, N, NRHS) For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (5*min(M, N)) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: the algorithm for computing the SVD failed to converge; if INFO = i, i off-diagonal elements of an intermediate bidiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *dgelss*, *sgelss* and *zgelss*. It also exists with a native C interface as *LAPACKE_cgelss*.

4.5.4 cgelsy

cgelsy computes the minimum-norm solution to a complex linear least squares problem:

```
minimize || A * X - B ||
```

using a complete orthogonal factorization of A. A is an M-by-N matrix which may be rank-deficient.

Several right hand side vectors b and solution vectors x can be handled in a single call; they are stored as the columns of the M-by-NRHS right hand side matrix B and the N-by-NRHS solution matrix X.

The routine first computes a QR factorization with column pivoting:

```
A * P = Q * [ R11 R12 ]
            [  0  R22 ]
```

with R11 defined as the largest leading submatrix whose estimated condition number is less than 1/RCOND. The order of R11, RANK, is the effective rank of A.

Then, R22 is considered to be negligible, and R12 is annihilated by unitary transformations from the right, arriving at the complete orthogonal factorization:

```
A * P = Q * [ T11 0 ] * Z
            [  0  0 ]
```

The minimum-norm solution is then

```
X = P * Z**H [ inv(T11)*Q1**H*B ]
             [         0        ]
```

where Q1 consists of the first RANK columns of Q.

This routine is basically identical to the original xGELSX except three differences:

```
o The permutation of matrix B (the right hand side) is faster and
  more simple.
o The call to the subroutine xGEQPF has been substituted by the
  the call to the subroutine xGEQP3. This subroutine is a Blas-3
  version of the QR factorization with column pivoting.
o Matrix B (the right hand side) is updated with Blas-3.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine cgelsy(M, N, NRHS, A, LDA, B, LDB, JPVT, RCOND, RANK, WORK, LWORK,
                  RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgelsy_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *nrhs, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, armpl_int_t *jpvt, const float *rcond,
             armpl_int_t *rank, armpl_singlecomplex_t *work,
             const armpl_int_t *lwork, float *rwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of matrices B and X. NRHS >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, A has been overwritten by details of its complete orthogonal factorization.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the M-by-NRHS right hand side matrix B. On exit, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M, N).

JPVT Input and output parameter.

JPVT is INTEGER array, dimension (N)

On entry, if JPVT(i) .ne. 0, the i-th column of A is permuted to the front of AP, otherwise column i is a free column. On exit, if JPVT(i) = k, then the i-th column of A*P was the k-th column of A.

RCOND Input parameter.

RCOND is REAL

RCOND is used to determine the effective rank of A, which is defined as the order of the largest leading triangular submatrix R11 in the QR factorization with pivoting of A, whose estimated condition number < 1/RCOND.

RANK Output parameter.

RANK is INTEGER

The effective rank of A, i.e., the order of the submatrix R11. This is the same as the order of the submatrix T11 in the complete orthogonal factorization of A.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. The unblocked strategy requires that: LWORK >= MN + MAX(2*MN, N+1, MN+NRHS) where MN = min(M, N). The block algorithm requires that: LWORK >= MN + MAX(2*MN, NB*(N+1), MN+MN*NB, MN+NB* NRHS) where NB is an upper bound on the blocksize returned by ILAENV for the routines CGEQP3, CTZRZF, CTZRQF, CUNMQR, and CUNMRZ.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dgelsy*, *sgelsy* and *zgelsy*. It also exists with a native C interface as *LAPACKE_cgelsy*.

4.5.5 cggglm

cggglm solves a general Gauss-Markov linear model (GLM) problem:

```
minimize || y ||_2   subject to   d = A*x + B*y
   x
```

where A is an N-by-M matrix, B is an N-by-P matrix, and d is a given N-vector. It is assumed that M <= N <= M+P, and

```
rank(A) = M    and    rank( A B ) = N.
```

Under these assumptions, the constrained equation is always consistent, and there is a unique solution x and a minimal 2-norm solution y, which is obtained using a generalized QR factorization of the matrices (A, B) given by

```
A = Q*(R),    B = Q*T*Z.
      (0)
```

In particular, if matrix B is square nonsingular, then the problem GLM is equivalent to the following weighted linear least squares problem

```
minimize || inv(B)*(d-A*x) ||_2
   x
```

where inv(B) denotes the inverse of B.

Syntax

Fortran specification:

```
use armpl_library

subroutine cggglm(N, M, P, A, LDA, B, LDB, D, X, Y, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cggglm_(const armpl_int_t *n, const armpl_int_t *m, const armpl_int_t *p,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             armpl_singlecomplex_t *b, const armpl_int_t *ldb,
             armpl_singlecomplex_t *d, armpl_singlecomplex_t *x,
             armpl_singlecomplex_t *y, armpl_singlecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The number of rows of the matrices A and B. N >= 0.

M Input parameter.

M is INTEGER

The number of columns of the matrix A. 0 <= M <= N.

P Input parameter.

P is INTEGER

The number of columns of the matrix B. P >= N-M.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, M). On entry, the N-by-M matrix A. On exit, the upper triangular part of the array A contains the M-by-M upper triangular matrix R.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, P). On entry, the N-by-P matrix B. On exit, if N <= P, the upper triangle of the subarray B(1:N,P-N+1:P) contains the N-by-N upper triangular matrix T; if N > P, the elements on and above the (N-P)th subdiagonal contain the N-by-P upper trapezoidal matrix T.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

D Input and output parameter.

D is COMPLEX

D is an array, dimension (N). On entry, D is the left hand side of the GLM equation. On exit, D is destroyed.

X Output parameter.

X is COMPLEX

X is an array, dimension (M) .

Y Output parameter.

Y is COMPLEX

Y is an array, dimension (P). On exit, X and Y are the solutions of the GLM problem.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,N+M+P). For optimum performance, LWORK >= M+min(N, P)+max(N, P)*NB, where NB is an upper bound for the optimal blocksizes for CGEQRF, CGERQF, CUNMQR and CUNMRQ.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. = 1: the upper triangular factor R associated with A in the generalized QR factorization of the pair (A, B) is singular, so that rank(A) < M; the least squares solution could not be computed. = 2: the bottom (N-M) by (N-M) part of the upper trapezoidal factor T associated with B in the generalized QR factorization of the pair (A, B) is singular, so that rank(A B) < N; the least squares solution could not be computed.

Related Information

For this routine in other precisions, please see *dggglm*, *sggglm* and *zggglm*. It also exists with a native C interface as *LAPACKE_cggglm*.

4.5.6 cgglse

cgglse solves the linear equality-constrained least squares (LSE) problem:

```
minimize || c - A*x ||_2   subject to   B*x = d
```

where A is an M-by-N matrix, B is a P-by-N matrix, c is a given M-vector, and d is a given P-vector. It is assumed that P <= N <= M+P, and

```
rank(B) = P and  rank( (A) ) = N.
                      ( (B) )
```

These conditions ensure that the LSE problem has a unique solution, which is obtained using a generalized RQ factorization of the matrices (B, A) given by

```
B = (0 R)*Q,    A = Z*T*Q.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine cgglse(M, N, P, A, LDA, B, LDB, C, D, X, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgglse_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *p,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             armpl_singlecomplex_t *b, const armpl_int_t *ldb,
             armpl_singlecomplex_t *c, armpl_singlecomplex_t *d,
```

(continues on next page)

(continued from previous page)

```
            armpl_singlecomplex_t *x, armpl_singlecomplex_t *work,
            const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrices A and B. N >= 0.

P Input parameter.

P is INTEGER

The number of rows of the matrix B. 0 <= P <= N <= M+P.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the min(M, N)-by-N upper trapezoidal matrix T.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, N). On entry, the P-by-N matrix B. On exit, the upper triangle of the subarray B(1:P,N-P+1:N) contains the P-by-P upper triangular matrix R.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, P).

C Input and output parameter.

C is COMPLEX

C is an array, dimension (M). On entry, C contains the right hand side vector for the least squares part of the LSE problem. On exit, the residual sum of squares for the solution is given by the sum of squares of elements N-P+1 to M of vector C.

D Input and output parameter.

D is COMPLEX

D is an array, dimension (P). On entry, D contains the right hand side vector for the constrained equation. On exit, D is destroyed.

X Output parameter.

X is COMPLEX

X is an array, dimension (N). On exit, X is the solution of the LSE problem.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,M+N+P). For optimum performance LWORK >= P+min(M, N)+max(M, N)*NB, where NB is an upper bound for the optimal blocksizes for CGEQRF, CGERQF, CUNMQR and CUNMRQ.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. = 1: the upper triangular factor R associated with B in the generalized RQ factorization of the pair (B, A) is singular, so that rank(B) < P; the least squares solution could not be computed. = 2: the (N-P) by (N-P) part of the upper trapezoidal factor T associated with A in the generalized RQ factorization of the pair (B, A) is singular, so that rank((A)) < N; the least squares solution could not ((B)) be computed.

Related Information

For this routine in other precisions, please see *dgglse*, *sgglse* and *zgglse*. It also exists with a native C interface as *LAPACKE_cgglse*.

4.5.7 dgels

dgels solves overdetermined or underdetermined real linear systems involving an M-by-N matrix A, or its transpose, using a QR or LQ factorization of A. It is assumed that A has full rank.

The following options are provided:

1. If TRANS = 'N' and m >= n: find the least squares solution of

```
an overdetermined system, i.e., solve the least squares problem
            minimize || B - A*X ||.
```

2. If TRANS = 'N' and m < n: find the minimum norm solution of

```
an underdetermined system A * X = B.
```

3. If TRANS = 'T' and m >= n: find the minimum norm solution of

```
an underdetermined system A**T * X = B.
```

4. If TRANS = 'T' and m < n: find the least squares solution of

```
an overdetermined system, i.e., solve the least squares problem
            minimize || B - A**T * X ||.
```

Several right hand side vectors b and solution vectors x can be handled in a single call; they are stored as the columns of the M-by-NRHS right hand side matrix B and the N-by-NRHS solution matrix X.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgels(TRANS, M, N, NRHS, A, LDA, B, LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgels_(const char *trans, const armpl_int_t *m, const armpl_int_t *n,
            const armpl_int_t *nrhs, double *a, const armpl_int_t *lda,
            double *b, const armpl_int_t *ldb, double *work,
            const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': the linear system involves A; = 'T': the linear system involves A^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >=0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, if M >= N, A is overwritten by details of its QR factorization as returned by DGEQRF; if M < N, A is overwritten by details of its LQ factorization as returned by DGELQF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the matrix B of right hand side vectors, stored columnwise; B is M-by-NRHS if TRANS = 'N', or N-by-NRHS if TRANS = 'T'. On exit, if INFO = 0, B is overwritten by the solution vectors, stored columnwise: if TRANS = 'N' and m >= n, rows 1 to n of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of elements N+1 to M in that column; if TRANS = 'N' and m < n, rows 1 to N of B contain the minimum norm solution vectors; if TRANS = 'T' and m >= n, rows 1 to M of B contain the minimum norm solution vectors; if TRANS = 'T' and m < n, rows 1 to M of B contain the least squares solution vectors; the

residual sum of squares for the solution in each column is given by the sum of squares of elements M+1 to N in that column.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= MAX(1, M, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, MN + max(MN, NRHS)). For optimal performance, LWORK >= max(1, MN + max(MN, NRHS)*NB). where MN = min(M, N) and NB is the optimum block size.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element of the triangular factor of A is zero, so that A does not have full rank; the least squares solution could not be computed.

Related Information

For this routine in other precisions, please see *cgels*, *sgels* and *zgels*. It also exists with a native C interface as *LAPACKE_dgels*.

4.5.8 dgelsd

dgelsd computes the minimum-norm solution to a real linear least squares problem:

```
minimize 2-norm(| b - A*x |)
```

using the singular value decomposition (SVD) of A. A is an M-by-N matrix which may be rank-deficient.

Several right hand side vectors b and solution vectors x can be handled in a single call; they are stored as the columns of the M-by-NRHS right hand side matrix B and the N-by-NRHS solution matrix X.

The problem is solved in three steps: (1) Reduce the coefficient matrix A to bidiagonal form with

```
Householder transformations, reducing the original problem
into a "bidiagonal least squares problem" (BLS)
```

(2) Solve the BLS using a divide and conquer approach.

(3) Apply back all the Householder transformations to solve

```
the original least squares problem.
```

The effective rank of A is determined by treating as zero those singular values which are less than RCOND times the largest singular value.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgelsd(M, N, NRHS, A, LDA, B, LDB, S, RCOND, RANK, WORK, LWORK,
                  IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgelsd_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *nrhs, const double *a, const armpl_int_t *lda,
             double *b, const armpl_int_t *ldb, double *s,
             const double *rcond, armpl_int_t *rank, double *work,
             const armpl_int_t *lwork, armpl_int_t *iwork,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, A has been destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the M-by-NRHS right hand side matrix B. On exit, B is overwritten by the N-by-NRHS solution matrix X. If m >= n and RANK = n, the residual sum-of-squares for the solution in the i-th column is given by the sum of squares of elements n+1:m in that column.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1,max(M, N)).

S Output parameter.

S is DOUBLE PRECISION

S is an array, dimension (min(M, N)). The singular values of A in decreasing order. The condition number of A in the 2-norm = S(1)/S(min(m,n)).

RCOND Input parameter.

RCOND is DOUBLE PRECISION

RCOND is used to determine the effective rank of A. Singular values S(i) <= RCOND*S(1) are treated as zero. If RCOND < 0, machine precision is used instead.

RANK Output parameter.

RANK is INTEGER

The effective rank of A, i.e., the number of singular values which are greater than RCOND*S(1).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK must be at least 1. The exact minimum amount of workspace needed depends on M, N and NRHS. As long as LWORK is at least 12* N + 2*N*SMLSIZ + 8*N*NLVL + N* NRHS + (SMLSIZ+1)**2, if M is greater than or equal to N or 12* M + 2*M*SMLSIZ + 8*M*NLVL + M* NRHS + (SMLSIZ+1)**2, if M is less than N, the code will execute correctly. SMLSIZ is returned by ILAENV and is equal to the maximum size of the subproblems at the bottom of the computation tree (usually about 25), and NLVL = MAX(0, INT(LOG_2(MIN(M,N)/(SMLSIZ+1))) + 1) For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1,LIWORK))

LIWORK >= max(1, 3 * MINMN * NLVL + 11 * MINMN), where MINMN = MIN(M,N). On exit, if INFO = 0, IWORK(1) returns the minimum LIWORK.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: the algorithm for computing the SVD failed to converge; if INFO = i, i off-diagonal elements of an intermediate bidiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *cgelsd*, *sgelsd* and *zgelsd*. It also exists with a native C interface as *LAPACKE_dgelsd*.

4.5.9 dgelss

`dgelss` computes the minimum norm solution to a real linear least squares problem:

Minimize 2-norm(| b - A*x |).

using the singular value decomposition (SVD) of A. A is an M-by-N matrix which may be rank-deficient.

Several right hand side vectors b and solution vectors x can be handled in a single call; they are stored as the columns of the M-by-NRHS right hand side matrix B and the N-by-NRHS solution matrix X.

The effective rank of A is determined by treating as zero those singular values which are less than RCOND times the largest singular value.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgelss(M, N, NRHS, A, LDA, B, LDB, S, RCOND, RANK, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dgelss_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *nrhs, double *a, const armpl_int_t *lda,
             double *b, const armpl_int_t *ldb, double *s,
             const double *rcond, armpl_int_t *rank, double *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

`M` is `INTEGER`

The number of rows of the matrix `A`. `M` >= 0.

N Input parameter.

`N` is `INTEGER`

The number of columns of the matrix `A`. `N` >= 0.

NRHS Input parameter.

`NRHS` is `INTEGER`

The number of right hand sides, i.e., the number of columns of the matrices `B` and X. `NRHS` >= 0.

A Input and output parameter.

`A` is `DOUBLE PRECISION`

`A` is an array, dimension (`LDA`, `N`). On entry, the M-by-N matrix `A`. On exit, the first min(m,n) rows of `A` are overwritten with its right singular vectors, stored rowwise.

LDA Input parameter.

`LDA` is `INTEGER`

The leading dimension of the array `A`. `LDA` >= max(1, `M`).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the M-by-NRHS right hand side matrix B. On exit, B is overwritten by the N-by-NRHS solution matrix X. If m >= n and RANK = n, the residual sum-of-squares for the solution in the i-th column is given by the sum of squares of elements n+1:m in that column.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1,max(M, N)).

S Output parameter.

S is DOUBLE PRECISION

S is an array, dimension (min(M, N)). The singular values of A in decreasing order. The condition number of A in the 2-norm = S(1)/S(min(m,n)).

RCOND Input parameter.

RCOND is DOUBLE PRECISION

RCOND is used to determine the effective rank of A. Singular values S(i) <= RCOND*S(1) are treated as zero. If RCOND < 0, machine precision is used instead.

RANK Output parameter.

RANK is INTEGER

The effective rank of A, i.e., the number of singular values which are greater than RCOND*S(1).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= 1, and also: LWORK >= 3*min(M, N) + max(2*min(M, N), max(M, N), NRHS) For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: the algorithm for computing the SVD failed to converge; if INFO = i, i off-diagonal elements of an intermediate bidiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *cgelss*, *sgelss* and *zgelss*. It also exists with a native C interface as *LAPACKE_dgelss*.

4.5.10 dgelsy

`dgelsy` computes the minimum-norm solution to a real linear least squares problem:

```
minimize || A * X - B ||
```

using a complete orthogonal factorization of A. A is an M-by-N matrix which may be rank-deficient.

Several right hand side vectors b and solution vectors x can be handled in a single call; they are stored as the columns of the M-by-NRHS right hand side matrix B and the N-by-NRHS solution matrix X.

The routine first computes a QR factorization with column pivoting:

```
A * P = Q * [ R11 R12 ]
            [  0  R22 ]
```

with R11 defined as the largest leading submatrix whose estimated condition number is less than 1/RCOND. The order of R11, RANK, is the effective rank of A.

Then, R22 is considered to be negligible, and R12 is annihilated by orthogonal transformations from the right, arriving at the complete orthogonal factorization:

```
A * P = Q * [ T11 0 ] * Z
            [  0  0 ]
```

The minimum-norm solution is then

```
X = P * Z**T [ inv(T11)*Q1**T*B ]
             [         0        ]
```

where Q1 consists of the first RANK columns of Q.

This routine is basically identical to the original xGELSX except three differences:

```
o The call to the subroutine xGEQPF has been substituted by the
  the call to the subroutine xGEQP3. This subroutine is a Blas-3
  version of the QR factorization with column pivoting.
o Matrix B (the right hand side) is updated with Blas-3.
o The permutation of matrix B (the right hand side) is faster and
  more simple.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dgelsy(M, N, NRHS, A, LDA, B, LDB, JPVT, RCOND, RANK, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dgelsy_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *nrhs, double *a, const armpl_int_t *lda,
             double *b, const armpl_int_t *ldb, armpl_int_t *jpvt,
             const double *rcond, armpl_int_t *rank, double *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of matrices B and X. NRHS >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, A has been overwritten by details of its complete orthogonal factorization.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the M-by-NRHS right hand side matrix B. On exit, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M, N).

JPVT Input and output parameter.

JPVT is INTEGER array, dimension (N)

On entry, if JPVT(i) .ne. 0, the i-th column of A is permuted to the front of AP, otherwise column i is a free column. On exit, if JPVT(i) = k, then the i-th column of AP was the k-th column of A.

RCOND Input parameter.

RCOND is DOUBLE PRECISION

RCOND is used to determine the effective rank of A, which is defined as the order of the largest leading triangular submatrix R11 in the QR factorization with pivoting of A, whose estimated condition number < 1/RCOND.

RANK Output parameter.

RANK is INTEGER

The effective rank of A, i.e., the order of the submatrix R11. This is the same as the order of the submatrix T11 in the complete orthogonal factorization of A.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. The unblocked strategy requires that: LWORK >= MAX(MN+3*N+1, 2*MN+NRHS), where MN = min(M, N). The block algorithm requires that: LWORK >= MAX(MN+2*N+NB*(N+1), 2*MN+NB* NRHS), where NB is an upper bound on the blocksize returned by ILAENV for the routines DGEQP3, DTZRZF, STZRQF, DORMQR, and DORMRZ.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: If INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cgelsy*, *sgelsy* and *zgelsy*. It also exists with a native C interface as *LAPACKE_dgelsy*.

4.5.11 dggglm

dggglm solves a general Gauss-Markov linear model (GLM) problem:

```
minimize || y ||_2   subject to   d = A*x + B*y
   x
```

where A is an N-by-M matrix, B is an N-by-P matrix, and d is a given N-vector. It is assumed that M <= N <= M+P, and

```
rank(A) = M    and    rank( A B ) = N.
```

Under these assumptions, the constrained equation is always consistent, and there is a unique solution x and a minimal 2-norm solution y, which is obtained using a generalized QR factorization of the matrices (A, B) given by

```
A = Q*(R),   B = Q*T*Z.
      (0)
```

In particular, if matrix B is square nonsingular, then the problem GLM is equivalent to the following weighted linear least squares problem

```
minimize || inv(B)*(d-A*x) ||_2
   x
```

where inv(B) denotes the inverse of B.

Syntax

Fortran specification:

```
use armpl_library

subroutine dggglm(N, M, P, A, LDA, B, LDB, D, X, Y, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dggglm_(const armpl_int_t *n, const armpl_int_t *m, const armpl_int_t *p,
             double *a, const armpl_int_t *lda, double *b,
             const armpl_int_t *ldb, double *d, double *x, double *y,
             double *work, const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The number of rows of the matrices A and B. N >= 0.

M Input parameter.

M is INTEGER

The number of columns of the matrix A. 0 <= M <= N.

P Input parameter.

P is INTEGER

The number of columns of the matrix B. P >= N-M.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, M). On entry, the N-by-M matrix A. On exit, the upper triangular part of the array A contains the M-by-M upper triangular matrix R.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, P). On entry, the N-by-P matrix B. On exit, if N <= P, the upper triangle of the subarray B(1:N,P-N+1:P) contains the N-by-N upper triangular matrix T; if N > P, the elements on and above the (N-P)th subdiagonal contain the N-by-P upper trapezoidal matrix T.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry, D is the left hand side of the GLM equation. On exit, D is destroyed.

X Output parameter.

X is DOUBLE PRECISION

X is an array, dimension (M) .

Y Output parameter.

Y is DOUBLE PRECISION

Y is an array, dimension (P). On exit, X and Y are the solutions of the GLM problem.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,N+M+P). For optimum performance, LWORK >= M+min(N, P)+max(N, P)*NB, where NB is an upper bound for the optimal blocksizes for DGEQRF, SGERQF, DORMQR and SORMRQ.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. = 1: the upper triangular factor R associated with A in the generalized QR factorization of the pair (A, B) is singular, so that rank(A) < M; the least squares solution could not be computed. = 2: the bottom (N-M) by (N-M) part of the upper trapezoidal factor T associated with B in the generalized QR factorization of the pair (A, B) is singular, so that rank(A B) < N; the least squares solution could not be computed.

Related Information

For this routine in other precisions, please see *cggglm*, *sggglm* and *zggglm*. It also exists with a native C interface as *LAPACKE_dggglm*.

4.5.12 dgglse

dgglse solves the linear equality-constrained least squares (LSE) problem:

```
minimize || c - A*x ||_2   subject to   B*x = d
```

where A is an M-by-N matrix, B is a P-by-N matrix, c is a given M-vector, and d is a given P-vector. It is assumed that P <= N <= M+P, and

```
rank(B) = P and  rank( (A) ) = N.
                       ( (B) )
```

These conditions ensure that the LSE problem has a unique solution, which is obtained using a generalized RQ factorization of the matrices (B, A) given by

```
B = (0 R)*Q,    A = Z*T*Q.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dgglse(M, N, P, A, LDA, B, LDB, C, D, X, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgglse_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *p,
             double *a, const armpl_int_t *lda, double *b,
             const armpl_int_t *ldb, double *c, double *d, double *x,
             double *work, const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrices A and B. N >= 0.

P Input parameter.

P is INTEGER

The number of rows of the matrix B. 0 <= P <= N <= M+P.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the min(M, N)-by-N upper trapezoidal matrix T.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). On entry, the P-by-N matrix B. On exit, the upper triangle of the subarray B(1:P,N-P+1:N) contains the P-by-P upper triangular matrix R.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, P).

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (M). On entry, C contains the right hand side vector for the least squares part of the LSE problem. On exit, the residual sum of squares for the solution is given by the sum of squares of elements N-P+1 to M of vector C.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (P). On entry, D contains the right hand side vector for the constrained equation. On exit, D is destroyed.

X Output parameter.

X is DOUBLE PRECISION

X is an array, dimension (N). On exit, X is the solution of the LSE problem.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,M+N+P). For optimum performance LWORK >= P+min(M, N)+max(M, N)*NB, where NB is an upper bound for the optimal blocksizes for DGEQRF, SGERQF, DORMQR and SORMRQ.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. = 1: the upper triangular factor R associated with B in the generalized RQ factorization of the pair (B, A) is singular, so that rank(B) < P; the least squares solution could not be computed. = 2: the (N-P) by (N-P) part of the upper trapezoidal factor T associated with A in the generalized RQ factorization of the pair (B, A) is singular, so that rank((A)) < N; the least squares solution could not ((B)) be computed.

Related Information

For this routine in other precisions, please see *cgglse*, *sgglse* and *zgglse*. It also exists with a native C interface as *LAPACKE_dgglse*.

4.5.13 sgels

sgels solves overdetermined or underdetermined real linear systems involving an M-by-N matrix A, or its transpose, using a QR or LQ factorization of A. It is assumed that A has full rank.

The following options are provided:

1. If TRANS = 'N' and m >= n: find the least squares solution of

```
an overdetermined system, i.e., solve the least squares problem
            minimize || B - A*X ||.
```

2. If TRANS = 'N' and m < n: find the minimum norm solution of

```
an underdetermined system A * X = B.
```

3. If TRANS = 'T' and m >= n: find the minimum norm solution of

```
an underdetermined system A**T * X = B.
```

4. If TRANS = 'T' and m < n: find the least squares solution of

```
an overdetermined system, i.e., solve the least squares problem
            minimize || B - A**T * X ||.
```

Several right hand side vectors b and solution vectors x can be handled in a single call; they are stored as the columns of the M-by-NRHS right hand side matrix B and the N-by-NRHS solution matrix X.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgels(TRANS, M, N, NRHS, A, LDA, B, LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgels_(const char *trans, const armpl_int_t *m, const armpl_int_t *n,
            const armpl_int_t *nrhs, float *a, const armpl_int_t *lda,
            float *b, const armpl_int_t *ldb, float *work,
            const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': the linear system involves A; = 'T': the linear system involves A^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >=0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, if M >= N, A is overwritten by details of its QR factorization as returned by SGEQRF; if M < N, A is overwritten by details of its LQ factorization as returned by SGELQF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the matrix B of right hand side vectors, stored columnwise; B is M-by-NRHS if TRANS = 'N', or N-by-NRHS if TRANS = 'T'. On exit, if INFO = 0, B is overwritten by the solution vectors, stored columnwise: if TRANS = 'N' and m >= n, rows 1 to n of B contain the least

squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of elements N+1 to M in that column; if TRANS = 'N' and m < n, rows 1 to N of B contain the minimum norm solution vectors; if TRANS = 'T' and m >= n, rows 1 to M of B contain the minimum norm solution vectors; if TRANS = 'T' and m < n, rows 1 to M of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of elements M+1 to N in that column.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= MAX(1, M, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, MN + max(MN, NRHS)). For optimal performance, LWORK >= max(1, MN + max(MN, NRHS)*NB). where MN = min(M, N) and NB is the optimum block size.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element of the triangular factor of A is zero, so that A does not have full rank; the least squares solution could not be computed.

Related Information

For this routine in other precisions, please see *cgels*, *dgels* and *zgels*. It also exists with a native C interface as *LAPACKE_sgels*.

4.5.14 sgelsd

sgelsd computes the minimum-norm solution to a real linear least squares problem:

```
minimize 2-norm(| b - A*x |)
```

using the singular value decomposition (SVD) of A. A is an M-by-N matrix which may be rank-deficient.

Several right hand side vectors b and solution vectors x can be handled in a single call; they are stored as the columns of the M-by-NRHS right hand side matrix B and the N-by-NRHS solution matrix X.

The problem is solved in three steps: (1) Reduce the coefficient matrix A to bidiagonal form with

```
Householder transformations, reducing the original problem
into a "bidiagonal least squares problem" (BLS)
```

(2) Solve the BLS using a divide and conquer approach.

(3) Apply back all the Householder transformations to solve

```
the original least squares problem.
```

The effective rank of A is determined by treating as zero those singular values which are less than RCOND times the largest singular value.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgelsd(M, N, NRHS, A, LDA, B, LDB, S, RCOND, RANK, WORK, LWORK,
                  IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgelsd_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *nrhs, const float *a, const armpl_int_t *lda,
             float *b, const armpl_int_t *ldb, float *s, const float *rcond,
             armpl_int_t *rank, float *work, const armpl_int_t *lwork,
             armpl_int_t *iwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, A has been destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the M-by-NRHS right hand side matrix B. On exit, B is overwritten by the N-by-NRHS solution matrix X. If m >= n and RANK = n, the residual sum-of-squares for the solution in the i-th column is given by the sum of squares of elements n+1:m in that column.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1,max(M, N)).

S Output parameter.

S is REAL

S is an array, dimension (min(M, N)). The singular values of A in decreasing order. The condition number of A in the 2-norm = S(1)/S(min(m,n)).

RCOND Input parameter.

RCOND is REAL

RCOND is used to determine the effective rank of A. Singular values S(i) <= RCOND*S(1) are treated as zero. If RCOND < 0, machine precision is used instead.

RANK Output parameter.

RANK is INTEGER

The effective rank of A, i.e., the number of singular values which are greater than RCOND*S(1).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK must be at least 1. The exact minimum amount of workspace needed depends on M, N and NRHS. As long as LWORK is at least 12* N + 2*N*SMLSIZ + 8*N*NLVL + N* NRHS + (SMLSIZ+1)**2, if M is greater than or equal to N or 12* M + 2*M*SMLSIZ + 8*M*NLVL + M* NRHS + (SMLSIZ+1)**2, if M is less than N, the code will execute correctly. SMLSIZ is returned by ILAENV and is equal to the maximum size of the subproblems at the bottom of the computation tree (usually about 25), and NLVL = MAX(0, INT(LOG_2(MIN(M,N)/(SMLSIZ+1))) + 1) For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the array WORK and the minimum size of the array IWORK, and returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1,LIWORK))

LIWORK >= max(1, 3*MINMN*NLVL + 11*MINMN), where MINMN = MIN(M,N). On exit, if INFO = 0, IWORK(1) returns the minimum LIWORK.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: the algorithm for computing the SVD failed to converge; if INFO = i, i off-diagonal elements of an intermediate bidiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *cgelsd*, *dgelsd* and *zgelsd*. It also exists with a native C interface as *LAPACKE_sgelsd*.

4.5.15 sgelss

`sgelss` computes the minimum norm solution to a real linear least squares problem:

Minimize 2-norm(| b - A*x |).

using the singular value decomposition (SVD) of A. A is an M-by-N matrix which may be rank-deficient.

Several right hand side vectors b and solution vectors x can be handled in a single call; they are stored as the columns of the M-by-NRHS right hand side matrix B and the N-by-NRHS solution matrix X.

The effective rank of A is determined by treating as zero those singular values which are less than RCOND times the largest singular value.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgelss(M, N, NRHS, A, LDA, B, LDB, S, RCOND, RANK, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void sgelss_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *nrhs, float *a, const armpl_int_t *lda,
             float *b, const armpl_int_t *ldb, float *s, const float *rcond,
             armpl_int_t *rank, float *work, const armpl_int_t *lwork,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the first min(m,n) rows of A are overwritten with its right singular vectors, stored rowwise.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the M-by-NRHS right hand side matrix B. On exit, B is overwritten by the N-by-NRHS solution matrix X. If m >= n and RANK = n, the residual sum-of-squares for the solution in the i-th column is given by the sum of squares of elements n+1:m in that column.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1,max(M, N)).

S Output parameter.

S is REAL

S is an array, dimension (min(M, N)). The singular values of A in decreasing order. The condition number of A in the 2-norm = S(1)/S(min(m,n)).

RCOND Input parameter.

RCOND is REAL

RCOND is used to determine the effective rank of A. Singular values S(i) <= RCOND*S(1) are treated as zero. If RCOND < 0, machine precision is used instead.

RANK Output parameter.

RANK is INTEGER

The effective rank of A, i.e., the number of singular values which are greater than RCOND*S(1).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= 1, and also: LWORK >= 3*min(M, N) + max(2*min(M, N), max(M, N), NRHS) For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: the algorithm for computing the SVD failed to converge; if INFO = i, i off-diagonal elements of an intermediate bidiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *cgelss*, *dgelss* and *zgelss*. It also exists with a native C interface as *LAPACKE_sgelss*.

4.5.16 sgelsy

`sgelsy` computes the minimum-norm solution to a real linear least squares problem:

```
minimize || A * X - B ||
```

using a complete orthogonal factorization of A. A is an M-by-N matrix which may be rank-deficient.

Several right hand side vectors b and solution vectors x can be handled in a single call; they are stored as the columns of the M-by-NRHS right hand side matrix B and the N-by-NRHS solution matrix X.

The routine first computes a QR factorization with column pivoting:

```
A * P = Q * [ R11 R12 ]
            [  0  R22 ]
```

with R11 defined as the largest leading submatrix whose estimated condition number is less than 1/RCOND. The order of R11, RANK, is the effective rank of A.

Then, R22 is considered to be negligible, and R12 is annihilated by orthogonal transformations from the right, arriving at the complete orthogonal factorization:

```
A * P = Q * [ T11 0 ] * Z
            [  0  0 ]
```

The minimum-norm solution is then

```
X = P * Z**T [ inv(T11)*Q1**T*B ]
             [         0        ]
```

where Q1 consists of the first RANK columns of Q.

This routine is basically identical to the original xGELSX except three differences:

```
o The call to the subroutine xGEQPF has been substituted by the
  the call to the subroutine xGEQP3. This subroutine is a Blas-3
  version of the QR factorization with column pivoting.
o Matrix B (the right hand side) is updated with Blas-3.
o The permutation of matrix B (the right hand side) is faster and
  more simple.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine sgelsy(M, N, NRHS, A, LDA, B, LDB, JPVT, RCOND, RANK, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void sgelsy_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *nrhs, float *a, const armpl_int_t *lda,
             float *b, const armpl_int_t *ldb, armpl_int_t *jpvt,
             const float *rcond, armpl_int_t *rank, float *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of matrices B and X. NRHS >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, A has been overwritten by details of its complete orthogonal factorization.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the M-by-NRHS right hand side matrix B. On exit, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M, N).

JPVT Input and output parameter.

JPVT is INTEGER array, dimension (N)

On entry, if JPVT(i) .ne. 0, the i-th column of A is permuted to the front of AP, otherwise column i is a free column. On exit, if JPVT(i) = k, then the i-th column of AP was the k-th column of A.

RCOND Input parameter.

RCOND is REAL

RCOND is used to determine the effective rank of A, which is defined as the order of the largest leading triangular submatrix R11 in the QR factorization with pivoting of A, whose estimated condition number < 1/RCOND.

RANK Output parameter.

RANK is INTEGER

The effective rank of A, i.e., the order of the submatrix R11. This is the same as the order of the submatrix T11 in the complete orthogonal factorization of A.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. The unblocked strategy requires that: LWORK >= MAX(MN+3*N+1, 2*MN+NRHS), where MN = min(M, N). The block algorithm requires that: LWORK >= MAX(MN+2*N+NB*(N+1), 2*MN+NB* NRHS), where NB is an upper bound on the blocksize returned by ILAENV for the routines SGEQP3, STZRZF, STZRQF, SORMQR, and SORMRZ.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: If INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cgelsy*, *dgelsy* and *zgelsy*. It also exists with a native C interface as *LAPACKE_sgelsy*.

4.5.17 sggglm

sggglm solves a general Gauss-Markov linear model (GLM) problem:

```
minimize || y ||_2   subject to   d = A*x + B*y
   x
```

where A is an N-by-M matrix, B is an N-by-P matrix, and d is a given N-vector. It is assumed that M <= N <= M+P, and

```
rank(A) = M    and    rank( A B ) = N.
```

Under these assumptions, the constrained equation is always consistent, and there is a unique solution x and a minimal 2-norm solution y, which is obtained using a generalized QR factorization of the matrices (A, B) given by

```
A = Q*(R),   B = Q*T*Z.
      (0)
```

In particular, if matrix B is square nonsingular, then the problem GLM is equivalent to the following weighted linear least squares problem

```
minimize || inv(B)*(d-A*x) ||_2
   x
```

where inv(B) denotes the inverse of B.

Syntax

Fortran specification:

```
use armpl_library

subroutine sggglm(N, M, P, A, LDA, B, LDB, D, X, Y, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sggglm_(const armpl_int_t *n, const armpl_int_t *m, const armpl_int_t *p,
             float *a, const armpl_int_t *lda, float *b,
             const armpl_int_t *ldb, float *d, float *x, float *y,
             float *work, const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The number of rows of the matrices A and B. N >= 0.

M Input parameter.

M is INTEGER

The number of columns of the matrix A. 0 <= M <= N.

P Input parameter.

P is INTEGER

The number of columns of the matrix B. P >= N-M.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, M). On entry, the N-by-M matrix A. On exit, the upper triangular part of the array A contains the M-by-M upper triangular matrix R.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, P). On entry, the N-by-P matrix B. On exit, if N <= P, the upper triangle of the subarray B(1:N,P-N+1:P) contains the N-by-N upper triangular matrix T; if N > P, the elements on and above the (N-P)th subdiagonal contain the N-by-P upper trapezoidal matrix T.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry, D is the left hand side of the GLM equation. On exit, D is destroyed.

X Output parameter.

X is REAL

X is an array, dimension (M) .

Y Output parameter.

Y is REAL

Y is an array, dimension (P). On exit, X and Y are the solutions of the GLM problem.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,N+M+P). For optimum performance, LWORK >= M+min(N, P)+max(N, P)*NB, where NB is an upper bound for the optimal blocksizes for SGEQRF, SGERQF, SORMQR and SORMRQ.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. = 1: the upper triangular factor R associated with A in the generalized QR factorization of the pair (A, B) is singular, so that rank(A) < M; the least squares solution could not be computed. = 2: the bottom (N-M) by (N-M) part of the upper trapezoidal factor T associated with B in the generalized QR factorization of the pair (A, B) is singular, so that rank(A B) < N; the least squares solution could not be computed.

Related Information

For this routine in other precisions, please see *cggglm*, *dggglm* and *zggglm*. It also exists with a native C interface as *LAPACKE_sggglm*.

4.5.18 sgglse

sgglse solves the linear equality-constrained least squares (LSE) problem:

```
minimize || c - A*x ||_2   subject to   B*x = d
```

where A is an M-by-N matrix, B is a P-by-N matrix, c is a given M-vector, and d is a given P-vector. It is assumed that P <= N <= M+P, and

```
rank(B) = P and  rank( (A) ) = N.
                      ( (B) )
```

These conditions ensure that the LSE problem has a unique solution, which is obtained using a generalized RQ factorization of the matrices (B, A) given by

```
B = (0 R)*Q,    A = Z*T*Q.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine sgglse(M, N, P, A, LDA, B, LDB, C, D, X, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgglse_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *p,
             float *a, const armpl_int_t *lda, float *b,
             const armpl_int_t *ldb, float *c, float *d, float *x,
             float *work, const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrices A and B. N >= 0.

P Input parameter.

P is INTEGER

The number of rows of the matrix B. 0 <= P <= N <= M+P.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the min(M, N)-by-N upper trapezoidal matrix T.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, N). On entry, the P-by-N matrix B. On exit, the upper triangle of the subarray B(1:P,N-P+1:N) contains the P-by-P upper triangular matrix R.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, P).

C Input and output parameter.

C is REAL

C is an array, dimension (M). On entry, C contains the right hand side vector for the least squares part of the LSE problem. On exit, the residual sum of squares for the solution is given by the sum of squares of elements N-P+1 to M of vector C.

D Input and output parameter.

D is REAL

D is an array, dimension (P). On entry, D contains the right hand side vector for the constrained equation. On exit, D is destroyed.

X Output parameter.

X is REAL

X is an array, dimension (N). On exit, X is the solution of the LSE problem.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,M+N+P). For optimum performance LWORK >= P+min(M, N)+max(M, N)*NB, where NB is an upper bound for the optimal blocksizes for SGEQRF, SGERQF, SORMQR and SORMRQ.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. = 1: the upper triangular factor R associated with B in the generalized RQ factorization of the pair (B, A) is singular, so that rank(B) < P; the least squares solution could not be computed. = 2: the (N-P) by (N-P) part of the upper trapezoidal factor T associated with A in the generalized RQ factorization of the pair (B, A) is singular, so that rank((A)) < N; the least squares solution could not ((B)) be computed.

Related Information

For this routine in other precisions, please see *cgglse*, *dgglse* and *zgglse*. It also exists with a native C interface as *LAPACKE_sgglse*.

4.5.19 zgels

zgels solves overdetermined or underdetermined complex linear systems involving an M-by-N matrix A, or its conjugate-transpose, using a QR or LQ factorization of A. It is assumed that A has full rank.

The following options are provided:

1. If TRANS = 'N' and m >= n: find the least squares solution of

```
an overdetermined system, i.e., solve the least squares problem
            minimize || B - A*X ||.
```

2. If TRANS = 'N' and m < n: find the minimum norm solution of

```
an underdetermined system A * X = B.
```

3. If TRANS = 'C' and m >= n: find the minimum norm solution of

```
an underdetermined system A**H * X = B.
```

4. If TRANS = 'C' and m < n: find the least squares solution of

```
an overdetermined system, i.e., solve the least squares problem
            minimize || B - A**H * X ||.
```

Several right hand side vectors b and solution vectors x can be handled in a single call; they are stored as the columns of the M-by-NRHS right hand side matrix B and the N-by-NRHS solution matrix X.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgels(TRANS, M, N, NRHS, A, LDA, B, LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgels_(const char *trans, const armpl_int_t *m, const armpl_int_t *n,
            const armpl_int_t *nrhs, armpl_doublecomplex_t *a,
            const armpl_int_t *lda, armpl_doublecomplex_t *b,
            const armpl_int_t *ldb, armpl_doublecomplex_t *work,
            const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': the linear system involves A; = 'C': the linear system involves A^H .

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. if M >= N, A is overwritten by details of its QR factorization as returned by ZGEQRF; if M < N, A is overwritten by details of its LQ factorization as returned by ZGELQF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the matrix B of right hand side vectors, stored columnwise; B is M-by-NRHS if TRANS = 'N', or N-by-NRHS if TRANS = 'C'. On exit, if INFO = 0, B is overwritten

by the solution vectors, stored columnwise: if TRANS = 'N' and m >= n, rows 1 to n of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of the modulus of elements N+1 to M in that column; if TRANS = 'N' and m < n, rows 1 to N of B contain the minimum norm solution vectors; if TRANS = 'C' and m >= n, rows 1 to M of B contain the minimum norm solution vectors; if TRANS = 'C' and m < n, rows 1 to M of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of the modulus of elements M+1 to N in that column.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= MAX(1, M, N).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, MN + max(MN, NRHS)). For optimal performance, LWORK >= max(1, MN + max(MN, NRHS)*NB). where MN = min(M, N) and NB is the optimum block size.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element of the triangular factor of A is zero, so that A does not have full rank; the least squares solution could not be computed.

Related Information

For this routine in other precisions, please see *cgels*, *dgels* and *sgels*. It also exists with a native C interface as *LAPACKE_zgels*.

4.5.20 zgelsd

zgelsd computes the minimum-norm solution to a real linear least squares problem:

```
minimize 2-norm(| b - A*x |)
```

using the singular value decomposition (SVD) of A. A is an M-by-N matrix which may be rank-deficient.

Several right hand side vectors b and solution vectors x can be handled in a single call; they are stored as the columns of the M-by-NRHS right hand side matrix B and the N-by-NRHS solution matrix X.

The problem is solved in three steps: (1) Reduce the coefficient matrix A to bidiagonal form with

```
Householder transformations, reducing the original problem
into a "bidiagonal least squares problem" (BLS)
```

(2) Solve the BLS using a divide and conquer approach.

(3) Apply back all the Householder transformations to solve

```
the original least squares problem.
```

The effective rank of A is determined by treating as zero those singular values which are less than RCOND times the largest singular value.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgelsd(M, N, NRHS, A, LDA, B, LDB, S, RCOND, RANK, WORK, LWORK,
                  RWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgelsd_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *nrhs, const armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, double *s, const double *rcond,
             armpl_int_t *rank, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, double *rwork, armpl_int_t *iwork,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, A has been destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the M-by-NRHS right hand side matrix B. On exit, B is overwritten by the N-by-NRHS solution matrix X. If m >= n and RANK = n, the residual sum-of-squares for the solution in the i-th column is given by the sum of squares of the modulus of elements n+1:m in that column.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M, N).

S Output parameter.

S is DOUBLE PRECISION

S is an array, dimension (min(M, N)). The singular values of A in decreasing order. The condition number of A in the 2-norm = S(1)/S(min(m,n)).

RCOND Input parameter.

RCOND is DOUBLE PRECISION

RCOND is used to determine the effective rank of A. Singular values S(i) <= RCOND*S(1) are treated as zero. If RCOND < 0, machine precision is used instead.

RANK Output parameter.

RANK is INTEGER

The effective rank of A, i.e., the number of singular values which are greater than RCOND*S(1).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK must be at least 1. The exact minimum amount of workspace needed depends on M, N and NRHS. As long as LWORK is at least 2* N + N*NRHS if M is greater than or equal to N or 2* M + M*NRHS if M is less than N, the code will execute correctly. For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the array WORK and the minimum sizes of the arrays RWORK and IWORK, and returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (MAX(1,LRWORK)). LRWORK >= 10* N + 2*N*SMLSIZ + 8*N*NLVL + 3*SMLSIZ* NRHS + MAX((SMLSIZ+1)**2, N*(1+NRHS) + 2* NRHS) if M is greater than or equal to N or 10* M + 2*M*SMLSIZ + 8*M*NLVL + 3*SMLSIZ* NRHS + MAX((SMLSIZ+1)**2, N*(1+NRHS) + 2* NRHS) if M is less than N, the code will execute correctly. SMLSIZ is returned by ILAENV and is equal to the maximum size of the subproblems at the bottom of the computation tree (usually about 25), and NLVL = MAX(0, INT(LOG_2(MIN(M,N)/(SMLSIZ+1))) + 1) On exit, if INFO = 0, RWORK(1) returns the minimum LRWORK.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1,LIWORK))

LIWORK >= max(1, 3*MINMN*NLVL + 11*MINMN), where MINMN = MIN(M,N). On exit, if INFO = 0, IWORK(1) returns the minimum LIWORK.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: the algorithm for computing the SVD failed to converge; if INFO = i, i off-diagonal elements of an intermediate bidiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *cgelsd*, *dgelsd* and *sgelsd*. It also exists with a native C interface as *LAPACKE_zgelsd*.

4.5.21 zgelss

zgelss computes the minimum norm solution to a complex linear least squares problem:

Minimize 2-norm(| b - A*x |).

using the singular value decomposition (SVD) of A. A is an M-by-N matrix which may be rank-deficient.

Several right hand side vectors b and solution vectors x can be handled in a single call; they are stored as the columns of the M-by-NRHS right hand side matrix B and the N-by-NRHS solution matrix X.

The effective rank of A is determined by treating as zero those singular values which are less than RCOND times the largest singular value.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgelss(M, N, NRHS, A, LDA, B, LDB, S, RCOND, RANK, WORK, LWORK,
                  RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgelss_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *nrhs, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, double *s, const double *rcond,
             armpl_int_t *rank, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, double *rwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the first min(m,n) rows of A are overwritten with its right singular vectors, stored rowwise.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the M-by-NRHS right hand side matrix B. On exit, B is overwritten by the N-by-NRHS solution matrix X. If m >= n and RANK = n, the residual sum-of-squares for the solution in the i-th column is given by the sum of squares of the modulus of elements n+1:m in that column.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M, N).

S Output parameter.

S is DOUBLE PRECISION

S is an array, dimension (min(M, N)). The singular values of A in decreasing order. The condition number of A in the 2-norm = S(1)/S(min(m,n)).

RCOND Input parameter.

RCOND is DOUBLE PRECISION

RCOND is used to determine the effective rank of A. Singular values S(i) <= RCOND*S(1) are treated as zero. If RCOND < 0, machine precision is used instead.

RANK Output parameter.

RANK is INTEGER

The effective rank of A, i.e., the number of singular values which are greater than RCOND*S(1).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= 1, and also: LWORK >= 2*min(M, N) + max(M, N, NRHS) For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (5*min(M, N)) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: the algorithm for computing the SVD failed to converge; if INFO = i, i off-diagonal elements of an intermediate bidiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *cgelss*, *dgelss* and *sgelss*. It also exists with a native C interface as *LAPACKE_zgelss*.

4.5.22 zgelsy

zgelsy computes the minimum-norm solution to a complex linear least squares problem:

```
minimize || A * X - B ||
```

using a complete orthogonal factorization of A. A is an M-by-N matrix which may be rank-deficient.

Several right hand side vectors b and solution vectors x can be handled in a single call; they are stored as the columns of the M-by-NRHS right hand side matrix B and the N-by-NRHS solution matrix X.

The routine first computes a QR factorization with column pivoting:

```
A * P = Q * [ R11 R12 ]
            [  0  R22 ]
```

with R11 defined as the largest leading submatrix whose estimated condition number is less than 1/RCOND. The order of R11, RANK, is the effective rank of A.

Then, R22 is considered to be negligible, and R12 is annihilated by unitary transformations from the right, arriving at the complete orthogonal factorization:

```
A * P = Q * [ T11 0 ] * Z
            [  0  0 ]
```

The minimum-norm solution is then

```
X = P * Z**H [ inv(T11)*Q1**H*B ]
             [         0        ]
```

where Q1 consists of the first RANK columns of Q.

This routine is basically identical to the original xGELSX except three differences:

```
o The permutation of matrix B (the right hand side) is faster and
  more simple.
o The call to the subroutine xGEQPF has been substituted by the
  the call to the subroutine xGEQP3. This subroutine is a Blas-3
  version of the QR factorization with column pivoting.
o Matrix B (the right hand side) is updated with Blas-3.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zgelsy(M, N, NRHS, A, LDA, B, LDB, JPVT, RCOND, RANK, WORK, LWORK,
                  RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgelsy_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *nrhs, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, armpl_int_t *jpvt, const double *rcond,
             armpl_int_t *rank, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, double *rwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of matrices B and X. NRHS >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, A has been overwritten by details of its complete orthogonal factorization.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the M-by-NRHS right hand side matrix B. On exit, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M, N).

JPVT Input and output parameter.

JPVT is INTEGER array, dimension (N)

On entry, if JPVT(i) .ne. 0, the i-th column of A is permuted to the front of AP, otherwise column i is a free column. On exit, if JPVT(i) = k, then the i-th column of A*P was the k-th column of A.

RCOND Input parameter.

RCOND is DOUBLE PRECISION

RCOND is used to determine the effective rank of A, which is defined as the order of the largest leading triangular submatrix R11 in the QR factorization with pivoting of A, whose estimated condition number < 1/RCOND.

RANK Output parameter.

RANK is INTEGER

The effective rank of A, i.e., the order of the submatrix R11. This is the same as the order of the submatrix T11 in the complete orthogonal factorization of A.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. The unblocked strategy requires that: LWORK >= MN + MAX(2*MN, N+1, MN+NRHS) where MN = min(M, N). The block algorithm requires that: LWORK >= MN + MAX(2*MN, NB*(N+1), MN+MN*NB, MN+NB* NRHS) where NB is an upper bound on the blocksize returned by ILAENV for the routines ZGEQP3, ZTZRZF, CTZRQF, ZUNMQR, and ZUNMRZ.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgelsy*, *dgelsy* and *sgelsy*. It also exists with a native C interface as *LAPACKE_zgelsy*.

4.5.23 zggglm

zggglm solves a general Gauss-Markov linear model (GLM) problem:

```
minimize || y ||_2   subject to   d = A*x + B*y
   x
```

where A is an N-by-M matrix, B is an N-by-P matrix, and d is a given N-vector. It is assumed that M <= N <= M+P, and

```
rank(A) = M    and    rank( A B ) = N.
```

Under these assumptions, the constrained equation is always consistent, and there is a unique solution x and a minimal 2-norm solution y, which is obtained using a generalized QR factorization of the matrices (A, B) given by

```
A = Q*(R),    B = Q*T*Z.
      (0)
```

In particular, if matrix B is square nonsingular, then the problem GLM is equivalent to the following weighted linear least squares problem

```
minimize || inv(B)*(d-A*x) ||_2
   x
```

where inv(B) denotes the inverse of B.

Syntax

Fortran specification:

```
use armpl_library

subroutine zggglm(N, M, P, A, LDA, B, LDB, D, X, Y, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zggglm_(const armpl_int_t *n, const armpl_int_t *m, const armpl_int_t *p,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_doublecomplex_t *b, const armpl_int_t *ldb,
             armpl_doublecomplex_t *d, armpl_doublecomplex_t *x,
             armpl_doublecomplex_t *y, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The number of rows of the matrices A and B. N >= 0.

M Input parameter.

M is INTEGER

The number of columns of the matrix A. 0 <= M <= N.

P Input parameter.

P is INTEGER

The number of columns of the matrix B. P >= N-M.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, M). On entry, the N-by-M matrix A. On exit, the upper triangular part of the array A contains the M-by-M upper triangular matrix R.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, P). On entry, the N-by-P matrix B. On exit, if N <= P, the upper triangle of the subarray B(1:N,P-N+1:P) contains the N-by-N upper triangular matrix T; if N > P, the elements on and above the (N-P)th subdiagonal contain the N-by-P upper trapezoidal matrix T.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

D Input and output parameter.

D is COMPLEX*16

D is an array, dimension (N). On entry, D is the left hand side of the GLM equation. On exit, D is destroyed.

X Output parameter.

X is COMPLEX*16

X is an array, dimension (M) .

Y Output parameter.

Y is COMPLEX*16

Y is an array, dimension (P). On exit, X and Y are the solutions of the GLM problem.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,N+M+P). For optimum performance, LWORK >= M+min(N, P)+max(N, P)*NB, where NB is an upper bound for the optimal blocksizes for ZGEQRF, ZGERQF, ZUNMQR and ZUNMRQ.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. = 1: the upper triangular factor R associated with A in the generalized QR factorization of the pair (A, B) is singular, so that rank(A) < M; the least squares solution could not be computed. = 2: the bottom (N-M) by (N-M) part of the upper trapezoidal factor T associated with B in the generalized QR factorization of the pair (A, B) is singular, so that rank(A B) < N; the least squares solution could not be computed.

Related Information

For this routine in other precisions, please see *cggglm*, *dggglm* and *sggglm*. It also exists with a native C interface as *LAPACKE_zggglm*.

4.5.24 zgglse

zgglse solves the linear equality-constrained least squares (LSE) problem:

```
minimize || c - A*x ||_2   subject to   B*x = d
```

where A is an M-by-N matrix, B is a P-by-N matrix, c is a given M-vector, and d is a given P-vector. It is assumed that P <= N <= M+P, and

```
rank(B) = P and  rank( (A) ) = N.
                      ( (B) )
```

These conditions ensure that the LSE problem has a unique solution, which is obtained using a generalized RQ factorization of the matrices (B, A) given by

```
B = (0 R)*Q,    A = Z*T*Q.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zgglse(M, N, P, A, LDA, B, LDB, C, D, X, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgglse_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *p,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_doublecomplex_t *b, const armpl_int_t *ldb,
             armpl_doublecomplex_t *c, armpl_doublecomplex_t *d,
             armpl_doublecomplex_t *x, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrices A and B. N >= 0.

P Input parameter.

P is INTEGER

The number of rows of the matrix B. 0 <= P <= N <= M+P.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the min(M, N)-by-N upper trapezoidal matrix T.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). On entry, the P-by-N matrix B. On exit, the upper triangle of the subarray B(1:P,N-P+1:N) contains the P-by-P upper triangular matrix R.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, P).

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (M). On entry, C contains the right hand side vector for the least squares part of the LSE problem. On exit, the residual sum of squares for the solution is given by the sum of squares of elements N-P+1 to M of vector C.

D Input and output parameter.

D is COMPLEX*16

D is an array, dimension (P). On entry, D contains the right hand side vector for the constrained equation. On exit, D is destroyed.

X Output parameter.

X is COMPLEX*16

X is an array, dimension (N). On exit, X is the solution of the LSE problem.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,M+N+P). For optimum performance LWORK >= P+min(M, N)+max(M, N)*NB, where NB is an upper bound for the optimal blocksizes for ZGEQRF, CGERQF, ZUNMQR and CUNMRQ.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. = 1: the upper triangular factor R associated with B in the generalized RQ factorization of the pair (B, A) is singular, so that rank(B) < P; the least squares solution could not be computed. = 2: the (N-P) by (N-P) part of the upper trapezoidal factor T associated with A in the generalized RQ factorization of the pair (B, A) is singular, so that rank((A)) < N; the least squares solution could not ((B)) be computed.

Related Information

For this routine in other precisions, please see *cgglse*, *dgglse* and *sgglse*. It also exists with a native C interface as *LAPACKE_zgglse*.

4.6 LAPACK linear equation solving routines

4.6.1 cgbsv

`cgbsv` computes the solution to a complex system of linear equations A * X = B, where A is a band matrix of order N with KL subdiagonals and KU superdiagonals, and X and B are N-by-NRHS matrices.

The LU decomposition with partial pivoting and row interchanges is used to factor A as A = L * U, where L is a product of permutation and unit lower triangular matrices with KL subdiagonals, and U is upper triangular with KL+KU superdiagonals. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgbsv(N, KL, KU, NRHS, AB, LDAB, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void cgbsv_(const armpl_int_t *n, const armpl_int_t *kl,
            const armpl_int_t *ku, const armpl_int_t *nrhs,
            armpl_singlecomplex_t *ab, const armpl_int_t *ldab,
            armpl_int_t *ipiv, armpl_singlecomplex_t *b,
            const armpl_int_t *ldb, armpl_int_t *info);
```

Parameters

N Input parameter.

`N` is `INTEGER`

The number of linear equations, i.e., the order of the matrix A. `N` >= 0.

KL Input parameter.

`KL` is `INTEGER`

The number of subdiagonals within the band of A. `KL` >= 0.

KU Input parameter.

`KU` is `INTEGER`

The number of superdiagonals within the band of A. `KU` >= 0.

NRHS Input parameter.

`NRHS` is `INTEGER`

The number of right hand sides, i.e., the number of columns of the matrix `B`. `NRHS` >= 0.

AB Input and output parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows KL+1 to 2*KL+KU+1; rows 1 to KL of the array need not be set. The j-th column of A is stored in the j-th column of the array AB as follows: AB(KL+KU+1+i-j,j) = A(i,j) for max(1,j-KU)<=i<=min(N,j+KL) On exit, details of the factorization: U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1. See below for further details.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= 2*KL+KU+1.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices that define the permutation matrix P; row i of the matrix was interchanged with row IPIV(i).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and the solution has not been computed.

Related Information

For this routine in other precisions, please see *dgbsv*, *sgbsv* and *zgbsv*. It also exists with a native C interface as *LAPACKE_cgbsv*.

4.6.2 cgbsvx

cgbsvx uses the LU factorization to compute the solution to a complex system of linear equations A * X = B, A^T * X = B, or A^H * X = B, where A is a band matrix of order N with KL subdiagonals and KU superdiagonals, and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed by this subroutine:
```

1. If FACT = 'E', real scaling factors are computed to equilibrate

```
the system:
   TRANS = 'N':  diag(R)*A*diag(C)      *inv(diag(C))*X = diag(R)*B
   TRANS = 'T': (diag(R)*A*diag(C))**T  *inv(diag(R))*X = diag(C)*B
   TRANS = 'C': (diag(R)*A*diag(C))**H  *inv(diag(R))*X = diag(C)*B
```

(continues on next page)

(continued from previous page)

```
Whether or not the system will be equilibrated depends on the
scaling of the matrix A, but if equilibration is used, A is
overwritten by diag(R)*A*diag(C) and B by diag(R)*B (if TRANS='N')
or diag(C)*B (if TRANS = 'T' or 'C').
```

2. If FACT = 'N' or 'E', the LU decomposition is used to factor the

```
matrix A (after equilibration if FACT = 'E') as
   A = L * U,
where L is a product of permutation and unit lower triangular
matrices with KL subdiagonals, and U is upper triangular with
KL+KU superdiagonals.
```

3. If some U(i,i)=0, so that U is exactly singular, then the routine

```
returns with INFO = i. Otherwise, the factored form of A is used
to estimate the condition number of the matrix A.  If the
reciprocal of the condition number is less than machine precision,
INFO = N+1 is returned as a warning, but the routine still goes on
to solve for X and compute error bounds as described below.
```

4. The system of equations is solved for X using the factored form

```
of A.
```

5. Iterative refinement is applied to improve the computed solution

```
matrix and calculate error bounds and backward error estimates
for it.
```

6. If equilibration was used, the matrix X is premultiplied by

```
diag(C) (if TRANS = 'N') or diag(R) (if TRANS = 'T' or 'C') so
that it solves the original system before equilibration.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine cgbsvx(FACT, TRANS, N, KL, KU, NRHS, AB, LDAB, AFB, LDAFB, IPIV,
                  EQUED, R, C, B, LDB, X, LDX, RCOND, FERR, BERR, WORK, RWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void cgbsvx_(const char *fact, const char *trans, const armpl_int_t *n,
             const armpl_int_t *kl, const armpl_int_t *ku,
             const armpl_int_t *nrhs, armpl_singlecomplex_t *ab,
             const armpl_int_t *ldab, armpl_singlecomplex_t *afb,
             const armpl_int_t *ldafb, armpl_int_t *ipiv, char *equed,
             float *r, float *c, armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, armpl_singlecomplex_t *x,
             const armpl_int_t *ldx, float *rcond, float *ferr, float *berr,
             armpl_singlecomplex_t *work, float *rwork, armpl_int_t *info,
             ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AFB and IPIV contain the factored form of A. If EQUED is not 'N', the matrix A has been equilibrated with scaling factors given by R and C. AB, AFB, and IPIV are not modified. = 'N': The matrix A will be copied to AFB and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AFB and factored.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations. = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose)

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AB Input and output parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(KU+1+i-j,j) = A(i,j) for max(1,j-KU)<=i<=min(N,j+kl)

If FACT = 'F' and EQUED is not 'N', then A must have been equilibrated by the scaling factors in R and/or C. AB is not modified if FACT = 'F' or 'N', or if FACT = 'E' and EQUED = 'N' on exit.

On exit, if EQUED .ne. 'N', A is scaled as follows: EQUED = 'R': A := diag(R) * A EQUED = 'C': A := A * diag(C) EQUED = 'B': A := diag(R) * A * diag(C).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KL+KU+1.

AFB Input and output parameter.

AFB is COMPLEX

AFB is an array, dimension (LDAFB, N). If FACT = 'F', then AFB is an input argument and on entry contains details of the LU factorization of the band matrix A, as computed by CGBTRF. U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1. If EQUED .ne. 'N', then AFB is the factored form of the equilibrated matrix A.

If FACT = 'N', then AFB is an output argument and on exit returns details of the LU factorization of A.

If FACT = 'E', then AFB is an output argument and on exit returns details of the LU factorization of the equilibrated matrix A (see the description of AB for the form of the equilibrated matrix).

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AFB. LDAFB >= 2*KL+KU+1.

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains the pivot indices from the factorization A = L*U as computed by CGBTRF; row i of the matrix was interchanged with row IPIV(i).

If FACT = 'N', then IPIV is an output argument and on exit contains the pivot indices from the factorization A = L*U of the original matrix A.

If FACT = 'E', then IPIV is an output argument and on exit contains the pivot indices from the factorization A = L*U of the equilibrated matrix A.

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'R': Row equilibration, i.e., A has been premultiplied by diag(R). = 'C': Column equilibration, i.e., A has been postmultiplied by diag(C). = 'B': Both row and column equilibration, i.e., A has been replaced by diag(R) * A * diag(C). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

R Input and output parameter.

R is REAL

R is an array, dimension (N). The row scale factors for A. If EQUED = 'R' or 'B', A is multiplied on the left by diag(R); if EQUED = 'N' or 'C', R is not accessed. R is an input argument if FACT = 'F'; otherwise, R is an output argument. If FACT = 'F' and EQUED = 'R' or 'B', each element of R must be positive.

C Input and output parameter.

C is REAL

C is an array, dimension (N). The column scale factors for A. If EQUED = 'C' or 'B', A is multiplied on the right by diag(C); if EQUED = 'N' or 'R', C is not accessed. C is an input argument if FACT = 'F'; otherwise, C is an output argument. If FACT = 'F' and EQUED = 'C' or 'B', each element of C must be positive.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, if EQUED = 'N', B is not modified; if TRANS = 'N' and EQUED = 'R' or 'B', B is overwritten by diag(R)*B; if TRANS = 'T' or 'C' and EQUED = 'C' or 'B', B is overwritten by diag(C)*B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is COMPLEX

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X to the original system of equations. Note that A and B are modified on exit if EQUED .ne. 'N', and the solution to the equilibrated system is inv(diag(C))* X if TRANS = 'N' and EQUED = 'C' or 'B', or inv(diag(R))* X if TRANS = 'T' or 'C' and EQUED = 'R' or 'B'.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

The estimate of the reciprocal condition number of the matrix A after equilibration (if done). If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N). On exit, RWORK(1) contains the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If RWORK(1) is much less than 1, then the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, condition estimator RCOND, and forward error bound FERR could be unreliable. If factorization fails with 0<INFO<=N, then RWORK(1) contains the reciprocal pivot growth factor for the leading INFO columns of A.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+1: U is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *dgbsvx*, *sgbsvx* and *zgbsvx*. It also exists with a native C interface as *LAPACKE_cgbsvx*.

4.6.3 cgbsvxx

```
CGBSVXX uses the LU factorization to compute the solution to a
   complex system of linear equations  A * X = B,  where A is an
   N-by-N matrix and X and B are N-by-NRHS matrices.

   If requested, both normwise and maximum componentwise error bounds
   are returned. CGBSVXX will return a solution with a tiny
   guaranteed error (O(eps) where eps is the working machine
   precision) unless the matrix is very ill-conditioned, in which
   case a warning is returned. Relevant condition numbers also are
   calculated and returned.

   CGBSVXX accepts user-provided factorizations and equilibration
   factors; see the definitions of the FACT and EQUED options.
   Solving with refinement and using a factorization from a previous
   CGBSVXX call will also produce a solution with either O(eps)
   errors or warnings, but we cannot make that claim for general
   user-provided factorizations and equilibration factors if they
   differ from what CGBSVXX would itself produce.

                                                                  The␣
↪following steps are performed:

   1. If FACT = 'E', real scaling factors are computed to equilibrate
   the system:

     TRANS = 'N':  diag(R)*A*diag(C)     *inv(diag(C))*X = diag(R)*B
     TRANS = 'T': (diag(R)*A*diag(C))**T *inv(diag(R))*X = diag(C)*B
     TRANS = 'C': (diag(R)*A*diag(C))**H *inv(diag(R))*X = diag(C)*B

   Whether or not the system will be equilibrated depends on the
   scaling of the matrix A, but if equilibration is used, A is
   overwritten by diag(R)*A*diag(C) and B by diag(R)*B (if TRANS='N')
   or diag(C)*B (if TRANS = 'T' or 'C').

   2. If FACT = 'N' or 'E', the LU decomposition is used to factor
   the matrix A (after equilibration if FACT = 'E') as

     A = P * L * U,

   where P is a permutation matrix, L is a unit lower triangular
   matrix, and U is upper triangular.

   3. If some U(i,i)=0, so that U is exactly singular, then the
   routine returns with INFO = i. Otherwise, the factored form of A
   is used to estimate the condition number of the matrix A (see
   argument RCOND). If the reciprocal of the condition number is less
   than machine precision, the routine still goes on to solve for X
   and compute error bounds as described below.

   4. The system of equations is solved for X using the factored form
   of A.

   5. By default (unless PARAMS(LA_LINRX_ITREF_I) is set to zero),
   the routine will use iterative refinement to try to get a small
   error and error bounds.  Refinement calculates the residual to at
   least twice the working precision.

   6. If equilibration was used, the matrix X is premultiplied by
   diag(C) (if TRANS = 'N') or diag(R) (if TRANS = 'T' or 'C') so
   that it solves the original system before equilibration.
↪    Some optional parameters are bundled in the PARAMS array.  These␣
```

(continues on next page)

(continued from previous page)

```
    settings determine how refinement is performed, but often the
    defaults are acceptable.  If the defaults are acceptable, users
    can pass NPARAMS = 0 which prevents the source code from accessing
    the PARAMS argument.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine cgbsvxx(FACT, TRANS, N, KL, KU, NRHS, AB, LDAB, AFB, LDAFB, IPIV,
                   EQUED, R, C, B, LDB, X, LDX, RCOND, RPVGRW, BERR,
                   N_ERR_BNDS, ERR_BNDS_NORM, ERR_BNDS_COMP, NPARAMS, PARAMS,
                   WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgbsvxx_(const char *fact, const char *trans, const armpl_int_t *n,
              const armpl_int_t *kl, const armpl_int_t *ku,
              const armpl_int_t *nrhs, armpl_singlecomplex_t *ab,
              const armpl_int_t *ldab, armpl_singlecomplex_t *afb,
              const armpl_int_t *ldafb, armpl_int_t *ipiv, char *equed,
              float *r, float *c, armpl_singlecomplex_t *b,
              const armpl_int_t *ldb, armpl_singlecomplex_t *x,
              const armpl_int_t *ldx, float *rcond, float *rpvgrw,
              float *berr, const armpl_int_t *n_err_bnds,
              float *err_bnds_norm, float *err_bnds_comp,
              const armpl_int_t *nparams, float *params,
              armpl_singlecomplex_t *work, float *rwork, armpl_int_t *info,
              ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AF and IPIV contain the factored form of A. If EQUED is not 'N', the matrix A has been equilibrated with scaling factors given by R and C. A, AF, and IPIV are not modified. = 'N': The matrix A will be copied to AF and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AF and factored.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate Transpose = Transpose)

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AB Input and output parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(KU+1+i-j,j) = A(i,j) for max(1,j-KU)<=i<=min(N,j+kl)

If FACT = 'F' and EQUED is not 'N', then AB must have been equilibrated by the scaling factors in R and/or C. AB is not modified if FACT = 'F' or 'N', or if FACT = 'E' and EQUED = 'N' on exit.

On exit, if EQUED .ne. 'N', A is scaled as follows: EQUED = 'R': A := diag(R) * A EQUED = 'C': A := A * diag(C) EQUED = 'B': A := diag(R) * A * diag(C).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KL+KU+1.

AFB Input and output parameter.

AFB is COMPLEX

AFB is an array, dimension (LDAFB, N). If FACT = 'F', then AFB is an input argument and on entry contains details of the LU factorization of the band matrix A, as computed by CGBTRF. U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1. If EQUED .ne. 'N', then AFB is the factored form of the equilibrated matrix A.

If FACT = 'N', then AF is an output argument and on exit returns the factors L and U from the factorization A = P*L*U of the original matrix A.

If FACT = 'E', then AF is an output argument and on exit returns the factors L and U from the factorization A = P*L*U of the equilibrated matrix A (see the description of A for the form of the equilibrated matrix).

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AFB. LDAFB >= 2*KL+KU+1.

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains the pivot indices from the factorization A = P*L*U as computed by SGETRF; row i of the matrix was interchanged with row IPIV(i).

If FACT = 'N', then IPIV is an output argument and on exit contains the pivot indices from the factorization A = P*L*U of the original matrix A.

If FACT = 'E', then IPIV is an output argument and on exit contains the pivot indices from the factorization A = P*L*U of the equilibrated matrix A.

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'R': Row equilibration, i.e., A has been premultiplied by diag(R). = 'C': Column equilibration, i.e., A has been postmultiplied by diag(C). = 'B': Both row and column equilibration, i.e., A has been replaced by diag(R) * A * diag(C). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

R Input and output parameter.

R is REAL

R is an array, dimension (N). The row scale factors for A. If EQUED = 'R' or 'B', A is multiplied on the left by diag(R); if EQUED = 'N' or 'C', R is not accessed. R is an input argument if FACT = 'F'; otherwise, R is an output argument. If FACT = 'F' and EQUED = 'R' or 'B', each element of R must be positive. If R is output, each element of R is a power of the radix. If R is input, each element of R should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

C Input and output parameter.

C is REAL

C is an array, dimension (N). The column scale factors for A. If EQUED = 'C' or 'B', A is multiplied on the right by diag(C); if EQUED = 'N' or 'R', C is not accessed. C is an input argument if FACT = 'F'; otherwise, C is an output argument. If FACT = 'F' and EQUED = 'C' or 'B', each element of C must be positive. If C is output, each element of C is a power of the radix. If C is input, each element of C should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if EQUED = 'N', B is not modified; if TRANS = 'N' and EQUED = 'R' or 'B', B is overwritten by diag(R)*B; if TRANS = 'T' or 'C' and EQUED = 'C' or 'B', B is overwritten by diag(C)*B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is COMPLEX

X is an array, dimension (LDX, NRHS). If INFO = 0, the N-by-NRHS solution matrix X to the original system of equations. Note that A and B are modified on exit if EQUED .ne. 'N', and the solution to the equilibrated system is inv(diag(C))* X if TRANS = 'N' and EQUED = 'C' or 'B', or inv(diag(R))* X if TRANS = 'T' or 'C' and EQUED = 'R' or 'B'.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

RPVGRW Output parameter.

RPVGRW is REAL

Reciprocal pivot growth. On exit, this contains the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If this is much less than 1, then the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, estimated condition numbers, and error bounds could be unreliable. If factorization fails with 0<INFO<=N, then this contains the reciprocal pivot growth factor for the leading INFO columns of A. In SGESVX, this quantity is returned in WORK(1).

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). Componentwise relative backward error. This is the componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

N_ERR_BNDS Input parameter.

N_ERR_BNDS is INTEGER

Number of error bounds to return for each right hand side and each type (normwise or componentwise). See ERR_BNDS_NORM and ERR_BNDS_COMP below.

ERR_BNDS_NORM Output parameter.

ERR_BNDS_NORM is REAL

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Output parameter.

ERR_BNDS_COMP is REAL

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:, N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

NPARAMS Input parameter.

NPARAMS is INTEGER

Specifies the number of parameters set in PARAMS. If .LE. 0, the PARAMS array is never referenced and default values are used.

PARAMS Input and output parameter.

PARAMS is REAL

PARAMS is an array, dimension NPARAMS. Specifies algorithm parameters. If an entry is .LT. 0.0, then that entry will be filled with default value used for that parameter. Only positions up to NPARAMS are accessed; defaults are used for higher-numbered parameters.

PARAMS(LA_LINRX_ITREF_I = 1) : Whether to perform iterative refinement or not. Default: 1.0 = 0.0 : No refinement is performed, and no error bounds are computed. = 1.0 : Use the double-precision refinement algorithm, possibly with doubled-single computations if the compilation environment does not support DOUBLE PRECISION. (other values are reserved for future use)

PARAMS(LA_LINRX_ITHRESH_I = 2) : Maximum number of residual computations allowed for refinement. Default: 10 Aggressive: Set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in err_bnds_norm and err_bnds_comp may no longer be trustworthy.

PARAMS(LA_LINRX_CWISE_I = 3) : Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Default: 1.0 (attempt componentwise convergence)

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. The solution to every right-hand side is guaranteed. < 0: If INFO = -i, the i-th argument had an illegal value > 0 and <= N: U(INFO, INFO) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+J: The solution corresponding to the Jth right-hand side is not guaranteed. The solutions corresponding to other right- hand sides K with K > J may not be guaranteed as well, but only the first such right-hand side is reported. If a small componentwise error is not requested (PARAMS(3)

= 0.0) then the Jth right-hand side is the first with a normwise error bound that is not guaranteed (the smallest J such that ERR_BNDS_NORM(J,1) = 0.0). By default (PARAMS(3) = 1.0) the Jth right-hand side is the first with either a normwise or componentwise error bound that is not guaranteed (the smallest J such that either ERR_BNDS_NORM(J,1) = 0.0 or ERR_BNDS_COMP(J,1) = 0.0). See the definition of ERR_BNDS_NORM(:,1) and ERR_BNDS_COMP(:,1). To get information about all of the right-hand sides check `ERR_BNDS_NORM` or `ERR_BNDS_COMP`.

Related Information

For this routine in other precisions, please see *dgbsvxx*, *sgbsvxx* and *zgbsvxx*. It also exists with a native C interface as *LAPACKE_cgbsvxx*.

4.6.4 cgbtrs

`cgbtrs` solves a system of linear equations

```
A * X = B,  A**T * X = B,  or  A**H * X = B
```

with a general band matrix A using the LU factorization computed by CGBTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgbtrs(TRANS, N, KL, KU, NRHS, AB, LDAB, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void cgbtrs_(const char *trans, const armpl_int_t *n, const armpl_int_t *kl,
             const armpl_int_t *ku, const armpl_int_t *nrhs,
             const armpl_singlecomplex_t *ab, const armpl_int_t *ldab,
             const armpl_int_t *ipiv, armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

`TRANS` is `CHARACTER*1`

Specifies the form of the system of equations. = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose)

N Input parameter.

`N` is `INTEGER`

The order of the matrix A. `N` >= 0.

KL Input parameter.

`KL` is `INTEGER`

The number of subdiagonals within the band of A. `KL` >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AB Input parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). Details of the LU factorization of the band matrix A, as computed by CGBTRF. U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= 2*KL+KU+1.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices; for 1 <= i <= N, row i of the matrix was interchanged with row IPIV(i).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dgbtrs*, *sgbtrs* and *zgbtrs*. It also exists with a native C interface as *LAPACKE_cgbtrs*.

4.6.5 cgesv

cgesv computes the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N matrix and X and B are N-by-NRHS matrices.

The LU decomposition with partial pivoting and row interchanges is used to factor A as

```
A = P * L * U,
```

where P is a permutation matrix, L is unit lower triangular, and U is upper triangular. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgesv(N, NRHS, A, LDA, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void cgesv_(const armpl_int_t *n, const armpl_int_t *nrhs,
            armpl_singlecomplex_t *a, const armpl_int_t *lda,
            armpl_int_t *ipiv, armpl_singlecomplex_t *b,
            const armpl_int_t *ldb, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the N-by-N coefficient matrix A. On exit, the factors L and U from the factorization A = P*L*U; the unit diagonal elements of L are not stored.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices that define the permutation matrix P; row i of the matrix was interchanged with row IPIV(i).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

Related Information

For this routine in other precisions, please see *dgesv*, *sgesv* and *zgesv*. It also exists with a native C interface as *LAPACKE_cgesv*.

4.6.6 cgesvx

`cgesvx` uses the LU factorization to compute the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N matrix and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed:
```

1. If FACT = 'E', real scaling factors are computed to equilibrate

```
 the system:
   TRANS = 'N':  diag(R)*A*diag(C)      *inv(diag(C))*X = diag(R)*B
   TRANS = 'T': (diag(R)*A*diag(C))**T *inv(diag(R))*X = diag(C)*B
   TRANS = 'C': (diag(R)*A*diag(C))**H *inv(diag(R))*X = diag(C)*B
Whether or not the system will be equilibrated depends on the
scaling of the matrix A, but if equilibration is used, A is
overwritten by diag(R)*A*diag(C) and B by diag(R)*B (if TRANS='N')
or diag(C)*B (if TRANS = 'T' or 'C').
```

2. If FACT = 'N' or 'E', the LU decomposition is used to factor the

```
 matrix A (after equilibration if FACT = 'E') as
   A = P * L * U,
where P is a permutation matrix, L is a unit lower triangular
matrix, and U is upper triangular.
```

3. If some U(i,i)=0, so that U is exactly singular, then the routine

```
 returns with INFO = i. Otherwise, the factored form of A is used
to estimate the condition number of the matrix A.  If the
reciprocal of the condition number is less than machine precision,
INFO = N+1 is returned as a warning, but the routine still goes on
to solve for X and compute error bounds as described below.
```

4. The system of equations is solved for X using the factored form

```
of A.
```

5. Iterative refinement is applied to improve the computed solution

```
 matrix and calculate error bounds and backward error estimates
for it.
```

6. If equilibration was used, the matrix X is premultiplied by

```
 diag(C) (if TRANS = 'N') or diag(R) (if TRANS = 'T' or 'C') so
that it solves the original system before equilibration.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine cgesvx(FACT, TRANS, N, NRHS, A, LDA, AF, LDAF, IPIV, EQUED, R, C,
                  B, LDB, X, LDX, RCOND, FERR, BERR, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgesvx_(const char *fact, const char *trans, const armpl_int_t *n,
             const armpl_int_t *nrhs, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_singlecomplex_t *af,
             const armpl_int_t *ldaf, armpl_int_t *ipiv, char *equed,
             float *r, float *c, armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, armpl_singlecomplex_t *x,
             const armpl_int_t *ldx, float *rcond, float *ferr, float *berr,
             armpl_singlecomplex_t *work, float *rwork, armpl_int_t *info,
             ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AF and IPIV contain the factored form of A. If EQUED is not 'N', the matrix A has been equilibrated with scaling factors given by R and C. A, AF, and IPIV are not modified. = 'N': The matrix A will be copied to AF and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AF and factored.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose)

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A. If FACT = 'F' and EQUED is not 'N', then A must have been equilibrated by the scaling factors in R and/or C. A is not modified if FACT = 'F' or 'N', or if FACT = 'E' and EQUED = 'N' on exit.

On exit, if EQUED .ne. 'N', A is scaled as follows: EQUED = 'R': A := diag(R) * A EQUED = 'C': A := A * diag(C) EQUED = 'B': A := diag(R) * A * diag(C).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input and output parameter.

AF is COMPLEX

AF is an array, dimension (LDAF, N). If FACT = 'F', then AF is an input argument and on entry contains the factors L and U from the factorization A = P*L*U as computed by CGETRF. If EQUED .ne. 'N', then AF is the factored form of the equilibrated matrix A.

If FACT = 'N', then AF is an output argument and on exit returns the factors L and U from the factorization A = P*L*U of the original matrix A.

If FACT = 'E', then AF is an output argument and on exit returns the factors L and U from the factorization A = P*L*U of the equilibrated matrix A (see the description of A for the form of the equilibrated matrix).

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains the pivot indices from the factorization A = P*L*U as computed by CGETRF; row i of the matrix was interchanged with row IPIV(i).

If FACT = 'N', then IPIV is an output argument and on exit contains the pivot indices from the factorization A = P*L*U of the original matrix A.

If FACT = 'E', then IPIV is an output argument and on exit contains the pivot indices from the factorization A = P*L*U of the equilibrated matrix A.

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'R': Row equilibration, i.e., A has been premultiplied by diag(R). = 'C': Column equilibration, i.e., A has been postmultiplied by diag(C). = 'B': Both row and column equilibration, i.e., A has been replaced by diag(R) * A * diag(C). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

R Input and output parameter.

R is REAL

R is an array, dimension (N). The row scale factors for A. If EQUED = 'R' or 'B', A is multiplied on the left by diag(R); if EQUED = 'N' or 'C', R is not accessed. R is an input argument if FACT = 'F'; otherwise, R is an output argument. If FACT = 'F' and EQUED = 'R' or 'B', each element of R must be positive.

C Input and output parameter.

C is REAL

C is an array, dimension (N). The column scale factors for A. If EQUED = 'C' or 'B', A is multiplied on the right by diag(C); if EQUED = 'N' or 'R', C is not accessed. C is an input argument if FACT = 'F'; otherwise, C is an output argument. If FACT = 'F' and EQUED = 'C' or 'B', each element of C must be positive.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if EQUED = 'N', B is not modified; if TRANS = 'N' and EQUED = 'R' or 'B', B is overwritten by diag(R)*B; if TRANS = 'T' or 'C' and EQUED = 'C' or 'B', B is overwritten by diag(C)*B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is COMPLEX

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X to the original system of equations. Note that A and B are modified on exit if EQUED .ne. 'N', and the solution to the equilibrated system is inv(diag(C))* X if TRANS = 'N' and EQUED = 'C' or 'B', or inv(diag(R))* X if TRANS = 'T' or 'C' and EQUED = 'R' or 'B'.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

The estimate of the reciprocal condition number of the matrix A after equilibration (if done). If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (2*N). On exit, RWORK(1) contains the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If RWORK(1) is much less than 1, then the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, condition estimator RCOND, and forward error bound FERR could be unreliable. If factorization fails with 0<INFO<=N, then RWORK(1) contains the reciprocal pivot growth factor for the leading INFO columns of A.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+1: U is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *dgesvx*, *sgesvx* and *zgesvx*. It also exists with a native C interface as *LAPACKE_cgesvx*.

4.6.7 cgesvxx

```
CGESVXX uses the LU factorization to compute the solution to a
   complex system of linear equations  A * X = B,  where A is an
   N-by-N matrix and X and B are N-by-NRHS matrices.

   If requested, both normwise and maximum componentwise error bounds
   are returned. CGESVXX will return a solution with a tiny
   guaranteed error (O(eps) where eps is the working machine
   precision) unless the matrix is very ill-conditioned, in which
   case a warning is returned. Relevant condition numbers also are
   calculated and returned.

   CGESVXX accepts user-provided factorizations and equilibration
   factors; see the definitions of the FACT and EQUED options.
   Solving with refinement and using a factorization from a previous
   CGESVXX call will also produce a solution with either O(eps)
   errors or warnings, but we cannot make that claim for general
   user-provided factorizations and equilibration factors if they
   differ from what CGESVXX would itself produce.

                                                                  The␣
↪following steps are performed:

   1. If FACT = 'E', real scaling factors are computed to equilibrate
   the system:

     TRANS = 'N':  diag(R)*A*diag(C)     *inv(diag(C))*X = diag(R)*B
     TRANS = 'T': (diag(R)*A*diag(C))**T *inv(diag(R))*X = diag(C)*B
     TRANS = 'C': (diag(R)*A*diag(C))**H *inv(diag(R))*X = diag(C)*B

   Whether or not the system will be equilibrated depends on the
   scaling of the matrix A, but if equilibration is used, A is
   overwritten by diag(R)*A*diag(C) and B by diag(R)*B (if TRANS='N')
   or diag(C)*B (if TRANS = 'T' or 'C').

   2. If FACT = 'N' or 'E', the LU decomposition is used to factor
   the matrix A (after equilibration if FACT = 'E') as

     A = P * L * U,

   where P is a permutation matrix, L is a unit lower triangular
   matrix, and U is upper triangular.

   3. If some U(i,i)=0, so that U is exactly singular, then the
   routine returns with INFO = i. Otherwise, the factored form of A
   is used to estimate the condition number of the matrix A (see
   argument RCOND). If the reciprocal of the condition number is less
   than machine precision, the routine still goes on to solve for X
   and compute error bounds as described below.

   4. The system of equations is solved for X using the factored form
   of A.

   5. By default (unless PARAMS(LA_LINRX_ITREF_I) is set to zero),
   the routine will use iterative refinement to try to get a small
```

(continues on next page)

(continued from previous page)

```
   error and error bounds.  Refinement calculates the residual to at
   least twice the working precision.

   6. If equilibration was used, the matrix X is premultiplied by
   diag(C) (if TRANS = 'N') or diag(R) (if TRANS = 'T' or 'C') so
   that it solves the original system before equilibration.
↪    Some optional parameters are bundled in the PARAMS array.  These
    settings determine how refinement is performed, but often the
    defaults are acceptable.  If the defaults are acceptable, users
    can pass NPARAMS = 0 which prevents the source code from accessing
    the PARAMS argument.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine cgesvxx(FACT, TRANS, N, NRHS, A, LDA, AF, LDAF, IPIV, EQUED, R, C,
                   B, LDB, X, LDX, RCOND, RPVGRW, BERR, N_ERR_BNDS,
                   ERR_BNDS_NORM, ERR_BNDS_COMP, NPARAMS, PARAMS, WORK, RWORK,
                   INFO)
```

C specification:

```
#include "armpl.h"

void cgesvxx_(const char *fact, const char *trans, const armpl_int_t *n,
              const armpl_int_t *nrhs, armpl_singlecomplex_t *a,
              const armpl_int_t *lda, armpl_singlecomplex_t *af,
              const armpl_int_t *ldaf, armpl_int_t *ipiv, char *equed,
              float *r, float *c, armpl_singlecomplex_t *b,
              const armpl_int_t *ldb, armpl_singlecomplex_t *x,
              const armpl_int_t *ldx, float *rcond, float *rpvgrw,
              float *berr, const armpl_int_t *n_err_bnds,
              float *err_bnds_norm, float *err_bnds_comp,
              const armpl_int_t *nparams, float *params,
              armpl_singlecomplex_t *work, float *rwork, armpl_int_t *info,
              ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AF and IPIV contain the factored form of A. If EQUED is not 'N', the matrix A has been equilibrated with scaling factors given by R and C. A, AF, and IPIV are not modified. = 'N': The matrix A will be copied to AF and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AF and factored.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate Transpose)

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A. If FACT = 'F' and EQUED is not 'N', then A must have been equilibrated by the scaling factors in R and/or C. A is not modified if FACT = 'F' or 'N', or if FACT = 'E' and EQUED = 'N' on exit.

On exit, if EQUED .ne. 'N', A is scaled as follows: EQUED = 'R': A := diag(R) * A EQUED = 'C': A := A * diag(C) EQUED = 'B': A := diag(R) * A * diag(C).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input and output parameter.

AF is COMPLEX

AF is an array, dimension (LDAF, N). If FACT = 'F', then AF is an input argument and on entry contains the factors L and U from the factorization A = P*L*U as computed by CGETRF. If EQUED .ne. 'N', then AF is the factored form of the equilibrated matrix A.

If FACT = 'N', then AF is an output argument and on exit returns the factors L and U from the factorization A = P*L*U of the original matrix A.

If FACT = 'E', then AF is an output argument and on exit returns the factors L and U from the factorization A = P*L*U of the equilibrated matrix A (see the description of A for the form of the equilibrated matrix).

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains the pivot indices from the factorization A = P*L*U as computed by CGETRF; row i of the matrix was interchanged with row IPIV(i).

If FACT = 'N', then IPIV is an output argument and on exit contains the pivot indices from the factorization A = P*L*U of the original matrix A.

If FACT = 'E', then IPIV is an output argument and on exit contains the pivot indices from the factorization A = P*L*U of the equilibrated matrix A.

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'R': Row equilibration, i.e., A has been premultiplied by diag(R). = 'C': Column equilibration, i.e., A has been postmultiplied by diag(C). = 'B': Both row and column equilibration, i.e., A has been replaced by diag(R) * A * diag(C). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

R Input and output parameter.

R is REAL

R is an array, dimension (N). The row scale factors for A. If EQUED = 'R' or 'B', A is multiplied on the left by diag(R); if EQUED = 'N' or 'C', R is not accessed. R is an input argument if FACT = 'F'; otherwise, R is an output argument. If FACT = 'F' and EQUED = 'R' or 'B', each element of R must be positive. If R is output, each element of R is a power of the radix. If R is input, each element of R should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

C Input and output parameter.

C is REAL

C is an array, dimension (N). The column scale factors for A. If EQUED = 'C' or 'B', A is multiplied on the right by diag(C); if EQUED = 'N' or 'R', C is not accessed. C is an input argument if FACT = 'F'; otherwise, C is an output argument. If FACT = 'F' and EQUED = 'C' or 'B', each element of C must be positive. If C is output, each element of C is a power of the radix. If C is input, each element of C should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if EQUED = 'N', B is not modified; if TRANS = 'N' and EQUED = 'R' or 'B', B is overwritten by diag(R)*B; if TRANS = 'T' or 'C' and EQUED = 'C' or 'B', B is overwritten by diag(C)*B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is COMPLEX

X is an array, dimension (LDX, NRHS). If INFO = 0, the N-by-NRHS solution matrix X to the original system of equations. Note that A and B are modified on exit if EQUED .ne. 'N', and the solution to the equilibrated system is inv(diag(C))* X if TRANS = 'N' and EQUED = 'C' or 'B', or inv(diag(R))* X if TRANS = 'T' or 'C' and EQUED = 'R' or 'B'.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

RPVGRW Output parameter.

RPVGRW is REAL

Reciprocal pivot growth. On exit, this contains the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If this is much less than 1, then the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, estimated condition numbers, and error bounds could be unreliable. If factorization fails with 0<INFO<=N, then this contains the reciprocal pivot growth factor for the leading INFO columns of A. In CGESVX, this quantity is returned in WORK(1).

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). Componentwise relative backward error. This is the componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

N_ERR_BNDS Input parameter.

N_ERR_BNDS is INTEGER

Number of error bounds to return for each right hand side and each type (normwise or componentwise). See ERR_BNDS_NORM and ERR_BNDS_COMP below.

ERR_BNDS_NORM Output parameter.

ERR_BNDS_NORM is REAL

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Output parameter.

ERR_BNDS_COMP is REAL

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:, N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 “Guaranteed” error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch(‘Epsilon’). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch(‘Epsilon’) to determine if the error estimate is “guaranteed”. These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

NPARAMS Input parameter.

NPARAMS is INTEGER

Specifies the number of parameters set in PARAMS. If .LE. 0, the PARAMS array is never referenced and default values are used.

PARAMS Input and output parameter.

PARAMS is REAL

PARAMS is an array, dimension NPARAMS. Specifies algorithm parameters. If an entry is .LT. 0.0, then that entry will be filled with default value used for that parameter. Only positions up to NPARAMS are accessed; defaults are used for higher-numbered parameters.

PARAMS(LA_LINRX_ITREF_I = 1) : Whether to perform iterative refinement or not. Default: 1.0 = 0.0 : No refinement is performed, and no error bounds are computed. = 1.0 : Use the double-precision refinement algorithm, possibly with doubled-single computations if the compilation environment does not support DOUBLE PRECISION. (other values are reserved for future use)

PARAMS(LA_LINRX_ITHRESH_I = 2) : Maximum number of residual computations allowed for refinement. Default: 10 Aggressive: Set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in err_bnds_norm and err_bnds_comp may no longer be trustworthy.

PARAMS(LA_LINRX_CWISE_I = 3) : Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Default: 1.0 (attempt componentwise convergence)

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. The solution to every right-hand side is guaranteed. < 0: If INFO = -i, the i-th argument had an illegal value > 0 and <= N: U(INFO, INFO) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+J: The solution corresponding to the Jth right-hand side is not guaranteed. The solutions corresponding to other right- hand sides K with K > J may not be guaranteed as well, but only the first such right-hand side is reported. If a small componentwise error is not requested (PARAMS(3) = 0.0) then the Jth right-hand side is the first with a normwise error bound that is not guaranteed (the smallest J such that ERR_BNDS_NORM(J,1) = 0.0). By default (PARAMS(3) = 1.0) the Jth right-hand side is the first with either a normwise or componentwise error bound that is not guaranteed (the smallest J such that either ERR_BNDS_NORM(J,1) = 0.0 or ERR_BNDS_COMP(J,1) = 0.0). See the definition of

ERR_BNDS_NORM(:,1) and ERR_BNDS_COMP(:,1). To get information about all of the right-hand sides check ERR_BNDS_NORM or ERR_BNDS_COMP.

Related Information

For this routine in other precisions, please see *dgesvxx*, *sgesvxx* and *zgesvxx*. It also exists with a native C interface as *LAPACKE_cgesvxx*.

4.6.8 cgetrs

cgetrs solves a system of linear equations

```
A * X = B,  A**T * X = B,  or  A**H * X = B
```

with a general N-by-N matrix A using the LU factorization computed by CGETRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgetrs(TRANS, N, NRHS, A, LDA, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void cgetrs_(const char *trans, const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const armpl_int_t *ipiv, armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose)

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The factors L and U from the factorization A = P*L*U as computed by CGETRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from CGETRF; for 1<=i<=N, row i of the matrix was interchanged with row IPIV(i).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dgetrs*, *sgetrs* and *zgetrs*. It also exists with a native C interface as *LAPACKE_cgetrs*.

4.6.9 cgtsv

cgtsv solves the equation

```
A*X = B,
```

where A is an N-by-N tridiagonal matrix, by Gaussian elimination with partial pivoting.

Note that the equation A^T *X = B may be solved by interchanging the order of the arguments DU and DL.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgtsv(N, NRHS, DL, D, DU, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void cgtsv_(const armpl_int_t *n, const armpl_int_t *nrhs,
            armpl_singlecomplex_t *dl, armpl_singlecomplex_t *d,
            armpl_singlecomplex_t *du, armpl_singlecomplex_t *b,
            const armpl_int_t *ldb, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

DL Input and output parameter.

DL is COMPLEX

DL is an array, dimension (N-1). On entry, DL must contain the (n-1) subdiagonal elements of A. On exit, DL is overwritten by the (n-2) elements of the second superdiagonal of the upper triangular matrix U from the LU factorization of A, in DL(1), . . . , DL(n-2).

D Input and output parameter.

D is COMPLEX

D is an array, dimension (N). On entry, D must contain the diagonal elements of A. On exit, D is overwritten by the n diagonal elements of U.

DU Input and output parameter.

DU is COMPLEX

DU is an array, dimension (N-1). On entry, DU must contain the (n-1) superdiagonal elements of A. On exit, DU is overwritten by the (n-1) elements of the first superdiagonal of U.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, U(i,i) is exactly zero, and the solution has not been computed. The factorization has not been completed unless i = N.

Related Information

For this routine in other precisions, please see *dgtsv*, *sgtsv* and *zgtsv*. It also exists with a native C interface as *LAPACKE_cgtsv*.

4.6.10 cgtsvx

cgtsvx uses the LU factorization to compute the solution to a complex system of linear equations A * X = B, A^T * X = B, or A^H * X = B, where A is a tridiagonal matrix of order N and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed:
```

1. If FACT = 'N', the LU decomposition is used to factor the matrix A

```
as A = L * U, where L is a product of permutation and unit lower
bidiagonal matrices and U is upper triangular with nonzeros in
only the main diagonal and first two superdiagonals.
```

2. If some U(i,i)=0, so that U is exactly singular, then the routine

```
returns with INFO = i. Otherwise, the factored form of A is used
to estimate the condition number of the matrix A.  If the
reciprocal of the condition number is less than machine precision,
INFO = N+1 is returned as a warning, but the routine still goes on
to solve for X and compute error bounds as described below.
```

3. The system of equations is solved for X using the factored form

```
of A.
```

4. Iterative refinement is applied to improve the computed solution

```
matrix and calculate error bounds and backward error estimates
for it.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine cgtsvx(FACT, TRANS, N, NRHS, DL, D, DU, DLF, DF, DUF, DU2, IPIV, B,
                  LDB, X, LDX, RCOND, FERR, BERR, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgtsvx_(const char *fact, const char *trans, const armpl_int_t *n,
             const armpl_int_t *nrhs, const armpl_singlecomplex_t *dl,
             const armpl_singlecomplex_t *d, const armpl_singlecomplex_t *du,
             armpl_singlecomplex_t *dlf, armpl_singlecomplex_t *df,
             armpl_singlecomplex_t *duf, armpl_singlecomplex_t *du2,
             armpl_int_t *ipiv, const armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, armpl_singlecomplex_t *x,
             const armpl_int_t *ldx, float *rcond, float *ferr, float *berr,
             armpl_singlecomplex_t *work, float *rwork, armpl_int_t *info,
             ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of A has been supplied on entry. = 'F': DLF, DF, DUF, DU2, and IPIV contain the factored form of A; DL, D, DU, DLF, DF, DUF, DU2 and IPIV will not be modified. = 'N': The matrix will be copied to DLF, DF, and DUF and factored.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose)

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

DL Input parameter.

DL is COMPLEX

DL is an array, dimension (N-1). The (n-1) subdiagonal elements of A.

D Input parameter.

D is COMPLEX

D is an array, dimension (N). The n diagonal elements of A.

DU Input parameter.

DU is COMPLEX

DU is an array, dimension (N-1). The (n-1) superdiagonal elements of A.

DLF Input and output parameter.

DLF is COMPLEX

DLF is an array, dimension (N-1). If FACT = 'F', then DLF is an input argument and on entry contains the (n-1) multipliers that define the matrix L from the LU factorization of A as computed by CGTTRF.

If FACT = 'N', then DLF is an output argument and on exit contains the (n-1) multipliers that define the matrix L from the LU factorization of A.

DF Input and output parameter.

DF is COMPLEX

DF is an array, dimension (N). If FACT = 'F', then DF is an input argument and on entry contains the n diagonal elements of the upper triangular matrix U from the LU factorization of A.

If FACT = 'N', then DF is an output argument and on exit contains the n diagonal elements of the upper triangular matrix U from the LU factorization of A.

DUF Input and output parameter.

DUF is COMPLEX

DUF is an array, dimension (N-1). If FACT = 'F', then DUF is an input argument and on entry contains the (n-1) elements of the first superdiagonal of U.

If FACT = 'N', then DUF is an output argument and on exit contains the (n-1) elements of the first superdiagonal of U.

DU2 Input and output parameter.

DU2 is COMPLEX

DU2 is an array, dimension (N-2). If FACT = 'F', then DU2 is an input argument and on entry contains the (n-2) elements of the second superdiagonal of U.

If FACT = 'N', then DU2 is an output argument and on exit contains the (n-2) elements of the second superdiagonal of U.

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains the pivot indices from the LU factorization of A as computed by CGTTRF.

If FACT = 'N', then IPIV is an output argument and on exit contains the pivot indices from the LU factorization of A; row i of the matrix was interchanged with row IPIV(i). IPIV(i) will always be either i or i+1; IPIV(i) = i indicates a row interchange was not required.

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). The N-by-NRHS right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is COMPLEX

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

The estimate of the reciprocal condition number of the matrix A. If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: U(i,i) is exactly zero. The factorization has not been completed unless i = N, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+1: U is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *dgtsvx*, *sgtsvx* and *zgtsvx*. It also exists with a native C interface as *LAPACKE_cgtsvx*.

4.6.11 cgttrs

cgttrs solves one of the systems of equations

```
A * X = B,  A**T * X = B,  or  A**H * X = B,
```

with a tridiagonal matrix A using the LU factorization computed by CGTTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgttrs(TRANS, N, NRHS, DL, D, DU, DU2, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void cgttrs_(const char *trans, const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_singlecomplex_t *dl, const armpl_singlecomplex_t *d,
             const armpl_singlecomplex_t *du,
             const armpl_singlecomplex_t *du2, const armpl_int_t *ipiv,
             armpl_singlecomplex_t *b, const armpl_int_t *ldb,
             armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations. = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose)

N Input parameter.

N is INTEGER

The order of the matrix A.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

DL Input parameter.

DL is COMPLEX

DL is an array, dimension (N-1). The (n-1) multipliers that define the matrix L from the LU factorization of A.

D Input parameter.

D is COMPLEX

D is an array, dimension (N). The n diagonal elements of the upper triangular matrix U from the LU factorization of A.

DU Input parameter.

DU is COMPLEX

DU is an array, dimension (N-1). The (n-1) elements of the first super-diagonal of U.

DU2 Input parameter.

DU2 is COMPLEX

DU2 is an array, dimension (N-2). The (n-2) elements of the second super-diagonal of U.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices; for 1 <= i <= n, row i of the matrix was interchanged with row IPIV(i). IPIV(i) will always be either i or i+1; IPIV(i) = i indicates a row interchange was not required.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the matrix of right hand side vectors B. On exit, B is overwritten by the solution vectors X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dgttrs*, *sgttrs* and *zgttrs*. It also exists with a native C interface as *LAPACKE_cgttrs*.

4.6.12 chesv

chesv computes the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N Hermitian matrix and X and B are N-by-NRHS matrices.

The diagonal pivoting method is used to factor A as

```
A = U * D * U**H,  if UPLO = 'U', or
A = L * D * L**H,  if UPLO = 'L',
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is Hermitian and block diagonal with 1-by-1 and 2-by-2 diagonal blocks. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine chesv(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void chesv_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
            armpl_singlecomplex_t *a, const armpl_int_t *lda,
            armpl_int_t *ipiv, armpl_singlecomplex_t *b,
            const armpl_int_t *ldb, armpl_singlecomplex_t *work,
            const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^H or A = L*D*L^H as computed by CHETRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D, as determined by CHETRF. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged, and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= 1, and for best performance LWORK >= max(1,N*NB), where NB is the optimal blocksize for CHETRF. for LWORK < N, TRS will be done with Level BLAS 2 for LWORK >= N, TRS will be done with Level BLAS 3

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, so the solution could not be computed.

Related Information

For this routine in other precisions, please see *zhesv*. It also exists with a native C interface as *LAPACKE_chesv*.

4.6.13 chesv_aa

CHESV_AA computes the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N Hermitian matrix and X and B are N-by-NRHS matrices.

Aasen's algorithm is used to factor A as

```
 A = U * T * U**H,  if UPLO = 'U', or
A = L * T * L**H,  if UPLO = 'L',
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and T is Hermitian and tridiagonal. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine chesv_aa(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void chesv_aa_(const char *uplo, const armpl_int_t *n,
               const armpl_int_t *nrhs, armpl_singlecomplex_t *a,
               const armpl_int_t *lda, armpl_int_t *ipiv,
               armpl_singlecomplex_t *b, const armpl_int_t *ldb,
               armpl_singlecomplex_t *work, const armpl_int_t *lwork,
               armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the tridiagonal matrix T and the multipliers used to obtain the factor U or L from the factorization A = U*T*U^H or A = L*T*L^H as computed by CHETRF_AA.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

On exit, it contains the details of the interchanges, i.e., the row and column k of A were interchanged with the row and column IPIV(k).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= MAX(1,2*N,3*N-2), and for best performance LWORK >= MAX(1,N*NB), where NB is the optimal blocksize for CHETRF.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, so the solution could not be computed.

Related Information

For this routine in other precisions, please see *zhesv_aa*. It also exists with a native C interface as *LAPACKE_chesv_aa*.

4.6.14 chesv_rk

CHESV_RK computes the solution to a complex system of linear equations A * X = B, where A is an N-by-N Hermitian matrix and X and B are N-by-NRHS matrices.

The bounded Bunch-Kaufman (rook) diagonal pivoting method is used to factor A as

```
A = P*U*D*(U**H)*(P**T),  if UPLO = 'U', or
A = P*L*D*(L**H)*(P**T),  if UPLO = 'L',
```

where U (or L) is unit upper (or lower) triangular matrix, U^H (or L^H) is the conjugate of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is Hermitian and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

CHETRF_RK is called to compute the factorization of a complex Hermitian matrix. The factored form of A is then used to solve the system of equations A * X = B by calling BLAS3 routine CHETRS_3.

Syntax

Fortran specification:

```
use armpl_library

subroutine chesv_rk(UPLO, N, NRHS, A, LDA, E, IPIV, B, LDB, WORK, LWORK,
                    INFO)
```

C specification:

```
#include "armpl.h"

void chesv_rk_(const char *uplo, const armpl_int_t *n,
               const armpl_int_t *nrhs, armpl_singlecomplex_t *a,
               const armpl_int_t *lda, armpl_singlecomplex_t *e,
               armpl_int_t *ipiv, armpl_singlecomplex_t *b,
               const armpl_int_t *ldb, armpl_singlecomplex_t *work,
               const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the Hermitian matrix A is stored: = 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U': the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced.

If UPLO = 'L': the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, diagonal of the block diagonal matrix D and factors U or L as computed by CHETRF_RK: a) ONLY diagonal elements of the Hermitian block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

For more info see the description of CHETRF_RK routine.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Output parameter.

E is COMPLEX

E is an array, dimension (N). On exit, contains the output computed by the factorization routine CHETRF_RK, i.e. the superdiagonal (or subdiagonal) elements of the Hermitian block diagonal matrix D with 1-by-1 or

2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) is set to 0; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) is set to 0.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is set to 0 in both UPLO = 'U' or UPLO = 'L' cases.

For more info see the description of CHETRF_RK routine.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D, as determined by CHETRF_RK.

For more info see the description of CHETRF_RK routine.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)).. Work array used in the factorization stage. On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= 1. For best performance of factorization stage LWORK >= max(1,N*NB), where NB is the optimal blocksize for CHETRF_RK.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array for factorization stage, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit

< 0: If INFO = -k, the k-th argument had an illegal value

> 0: If INFO = k, the matrix A is singular, because: If UPLO = 'U': column k in the upper triangular part of A contains all zeros. If UPLO = 'L': column k in the lower triangular part of A contains all zeros.

Therefore D(k,k) is exactly zero, and superdiagonal elements of column k of U (or subdiagonal elements of column k of L) are all zeros. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

NOTE: INFO only stores the first occurrence of a singularity, any subsequent occurrence of singularity is not stored in INFO even though the factorization always completes.

Related Information

For this routine in other precisions, please see *zhesv_rk*. It also exists with a native C interface as *LAPACKE_chesv_rk*.

4.6.15 chesv_rook

CHESV_ROOK computes the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N Hermitian matrix and X and B are N-by-NRHS matrices.

The bounded Bunch-Kaufman ("rook") diagonal pivoting method is used to factor A as

```
A = U * D * U**T,  if UPLO = 'U', or
A = L * D * L**T,  if UPLO = 'L',
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is Hermitian and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

CHETRF_ROOK is called to compute the factorization of a complex Hermition matrix A using the bounded Bunch-Kaufman ("rook") diagonal pivoting method.

The factored form of A is then used to solve the system of equations A * X = B by calling CHETRS_ROOK (uses BLAS 2).

Syntax

Fortran specification:

```
use armpl_library

subroutine chesv_rook(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void chesv_rook_(const char *uplo, const armpl_int_t *n,
                 const armpl_int_t *nrhs, armpl_singlecomplex_t *a,
                 const armpl_int_t *lda, armpl_int_t *ipiv,
                 armpl_singlecomplex_t *b, const armpl_int_t *ldb,
                 armpl_singlecomplex_t *work, const armpl_int_t *lwork,
                 armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^H or A = L*D*L^H as computed by CHETRF_ROOK.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D.

If UPLO = 'U': Only the last KB elements of IPIV are set.

If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k-1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k-1 and -IPIV(k-1) were inerchaged, D(k-1:k,k-1:k) is a 2-by-2 diagonal block.

If UPLO = 'L': Only the first KB elements of IPIV are set.

If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k+1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k+1 and -IPIV(k+1) were inerchaged, D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= 1, and for best performance LWORK >= max(1,N*NB), where NB is the optimal blocksize for CHETRF_ROOK. for LWORK < N, TRS will be done with Level BLAS 2 for LWORK >= N, TRS will be done with Level BLAS 3

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, so the solution could not be computed.

Related Information

For this routine in other precisions, please see *zhesv_rook*.

4.6.16 chesvx

chesvx uses the diagonal pivoting factorization to compute the solution to a complex system of linear equations A * X = B, where A is an N-by-N Hermitian matrix and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed:
```

1. If FACT = 'N', the diagonal pivoting method is used to factor A.

```
 The form of the factorization is
   A = U * D * U**H,  if UPLO = 'U', or
   A = L * D * L**H,  if UPLO = 'L',
where U (or L) is a product of permutation and unit upper (lower)
triangular matrices, and D is Hermitian and block diagonal with
1-by-1 and 2-by-2 diagonal blocks.
```

2. If some D(i,i)=0, so that D is exactly singular, then the routine

```
 returns with INFO = i. Otherwise, the factored form of A is used
to estimate the condition number of the matrix A.  If the
reciprocal of the condition number is less than machine precision,
INFO = N+1 is returned as a warning, but the routine still goes on
to solve for X and compute error bounds as described below.
```

3. The system of equations is solved for X using the factored form

```
of A.
```

4. Iterative refinement is applied to improve the computed solution

```
 matrix and calculate error bounds and backward error estimates
for it.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine chesvx(FACT, UPLO, N, NRHS, A, LDA, AF, LDAF, IPIV, B, LDB, X, LDX,
                  RCOND, FERR, BERR, WORK, LWORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void chesvx_(const char *fact, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *nrhs, const armpl_singlecomplex_t *a,
```

(continues on next page)

(continued from previous page)

```
                const armpl_int_t *lda, armpl_singlecomplex_t *af,
                const armpl_int_t *ldaf, armpl_int_t *ipiv,
                const armpl_singlecomplex_t *b, const armpl_int_t *ldb,
                armpl_singlecomplex_t *x, const armpl_int_t *ldx, float *rcond,
                float *ferr, float *berr, armpl_singlecomplex_t *work,
                const armpl_int_t *lwork, float *rwork, armpl_int_t *info,
                ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of A has been supplied on entry. = 'F': On entry, AF and IPIV contain the factored form of A. A, AF and IPIV will not be modified. = 'N': The matrix A will be copied to AF and factored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input and output parameter.

AF is COMPLEX

AF is an array, dimension (LDAF, N). If FACT = 'F', then AF is an input argument and on entry contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^H or A = L*D*L^H as computed by CHETRF.

If FACT = 'N', then AF is an output argument and on exit returns the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^H or A = L*D*L^H .

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains details of the interchanges and the block structure of D, as determined by CHETRF. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

If FACT = 'N', then IPIV is an output argument and on exit contains details of the interchanges and the block structure of D, as determined by CHETRF.

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). The N-by-NRHS right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is COMPLEX

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

The estimate of the reciprocal condition number of the matrix A. If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= max(1,2*N), and for best performance, when FACT = 'N', LWORK >= max(1,2*N,N*NB), where NB is the optimal blocksize for CHETRF.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: D(i,i) is exactly zero. The factorization has been completed but the factor D is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+1: D is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *zhesvx*. It also exists with a native C interface as *LAPACKE_chesvx*.

4.6.17 chesvxx

```
CHESVXX uses the diagonal pivoting factorization to compute the
   solution to a complex system of linear equations A * X = B, where
   A is an N-by-N symmetric matrix and X and B are N-by-NRHS
   matrices.

   If requested, both normwise and maximum componentwise error bounds
   are returned. CHESVXX will return a solution with a tiny
   guaranteed error (O(eps) where eps is the working machine
   precision) unless the matrix is very ill-conditioned, in which
   case a warning is returned. Relevant condition numbers also are
   calculated and returned.

   CHESVXX accepts user-provided factorizations and equilibration
   factors; see the definitions of the FACT and EQUED options.
   Solving with refinement and using a factorization from a previous
   CHESVXX call will also produce a solution with either O(eps)
   errors or warnings, but we cannot make that claim for general
   user-provided factorizations and equilibration factors if they
   differ from what CHESVXX would itself produce.

                                                                  The␣
↪following steps are performed:

   1. If FACT = 'E', real scaling factors are computed to equilibrate
   the system:

     diag(S)*A*diag(S)      *inv(diag(S))*X = diag(S)*B
```

(continues on next page)

(continued from previous page)

```
   Whether or not the system will be equilibrated depends on the
   scaling of the matrix A, but if equilibration is used, A is
   overwritten by diag(S)*A*diag(S) and B by diag(S)*B.

   2. If FACT = 'N' or 'E', the LU decomposition is used to factor
   the matrix A (after equilibration if FACT = 'E') as

      A = U * D * U**T,  if UPLO = 'U', or
      A = L * D * L**T,  if UPLO = 'L',

   where U (or L) is a product of permutation and unit upper (lower)
   triangular matrices, and D is symmetric and block diagonal with
   1-by-1 and 2-by-2 diagonal blocks.

   3. If some D(i,i)=0, so that D is exactly singular, then the
   routine returns with INFO = i. Otherwise, the factored form of A
   is used to estimate the condition number of the matrix A (see
   argument RCOND).  If the reciprocal of the condition number is
   less than machine precision, the routine still goes on to solve
   for X and compute error bounds as described below.

   4. The system of equations is solved for X using the factored form
   of A.

   5. By default (unless PARAMS(LA_LINRX_ITREF_I) is set to zero),
   the routine will use iterative refinement to try to get a small
   error and error bounds.  Refinement calculates the residual to at
   least twice the working precision.

   6. If equilibration was used, the matrix X is premultiplied by
   diag(R) so that it solves the original system before
   equilibration.                          Some optional parameters are bundled␣
↪in the PARAMS array.  These
    settings determine how refinement is performed, but often the
    defaults are acceptable.  If the defaults are acceptable, users
    can pass NPARAMS = 0 which prevents the source code from accessing
    the PARAMS argument.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine chesvxx(FACT, UPLO, N, NRHS, A, LDA, AF, LDAF, IPIV, EQUED, S, B,
                   LDB, X, LDX, RCOND, RPVGRW, BERR, N_ERR_BNDS,
                   ERR_BNDS_NORM, ERR_BNDS_COMP, NPARAMS, PARAMS, WORK, RWORK,
                   INFO)
```

C specification:

```
#include "armpl.h"

void chesvxx_(const char *fact, char *uplo, const armpl_int_t *n,
              const armpl_int_t *nrhs, armpl_singlecomplex_t *a,
              const armpl_int_t *lda, armpl_singlecomplex_t *af,
              const armpl_int_t *ldaf, armpl_int_t *ipiv, char *equed,
              float *s, armpl_singlecomplex_t *b, const armpl_int_t *ldb,
              armpl_singlecomplex_t *x, const armpl_int_t *ldx, float *rcond,
              float *rpvgrw, float *berr, const armpl_int_t *n_err_bnds,
```

(continues on next page)

(continued from previous page)

```
                float *err_bnds_norm, float *err_bnds_comp,
                const armpl_int_t *nparams, float *params,
                armpl_singlecomplex_t *work, float *rwork, armpl_int_t *info,
                ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AF and IPIV contain the factored form of A. If EQUED is not 'N', the matrix A has been equilibrated with scaling factors given by S. A, AF, and IPIV are not modified. = 'N': The matrix A will be copied to AF and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AF and factored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if FACT = 'E' and EQUED = 'Y', A is overwritten by diag(S)*A*diag(S).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input and output parameter.

AF is COMPLEX

AF is an array, dimension (LDAF, N). If FACT = 'F', then AF is an input argument and on entry contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by SSYTRF.

If FACT = 'N', then AF is an output argument and on exit returns the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T .

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains details of the interchanges and the block structure of D, as determined by CHETRF. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

If FACT = 'N', then IPIV is an output argument and on exit contains details of the interchanges and the block structure of D, as determined by CHETRF.

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'Y': Both row and column equilibration, i.e., A has been replaced by diag(S) * A * diag(S). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

S Input and output parameter.

S is REAL

S is an array, dimension (N). The scale factors for A. If EQUED = 'Y', A is multiplied on the left and right by diag(S). S is an input argument if FACT = 'F'; otherwise, S is an output argument. If FACT = 'F' and EQUED = 'Y', each element of S must be positive. If S is output, each element of S is a power of the radix. If S is input, each element of S should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if EQUED = 'N', B is not modified; if EQUED = 'Y', B is overwritten by diag(S)*B;

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is COMPLEX

X is an array, dimension (LDX, NRHS). If INFO = 0, the N-by-NRHS solution matrix X to the original system of equations. Note that A and B are modified on exit if EQUED .ne. 'N', and the solution to the equilibrated system is inv(diag(S))*X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

RPVGRW Output parameter.

RPVGRW is REAL

Reciprocal pivot growth. On exit, this contains the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If this is much less than 1, then the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, estimated condition numbers, and error bounds could be unreliable. If factorization fails with 0<INFO<=N, then this contains the reciprocal pivot growth factor for the leading INFO columns of A.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). Componentwise relative backward error. This is the componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

N_ERR_BNDS Input parameter.

N_ERR_BNDS is INTEGER

Number of error bounds to return for each right hand side and each type (normwise or componentwise). See ERR_BNDS_NORM and ERR_BNDS_COMP below.

ERR_BNDS_NORM Output parameter.

ERR_BNDS_NORM is REAL

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Output parameter.

ERR_BNDS_COMP is REAL

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then

ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:, N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

NPARAMS Input parameter.

NPARAMS is INTEGER

Specifies the number of parameters set in PARAMS. If .LE. 0, the PARAMS array is never referenced and default values are used.

PARAMS Input and output parameter.

PARAMS is REAL

PARAMS is an array, dimension NPARAMS. Specifies algorithm parameters. If an entry is .LT. 0.0, then that entry will be filled with default value used for that parameter. Only positions up to NPARAMS are accessed; defaults are used for higher-numbered parameters.

PARAMS(LA_LINRX_ITREF_I = 1) : Whether to perform iterative refinement or not. Default: 1.0 = 0.0 : No refinement is performed, and no error bounds are computed. = 1.0 : Use the double-precision refinement algorithm, possibly with doubled-single computations if the compilation environment does not support DOUBLE PRECISION. (other values are reserved for future use)

PARAMS(LA_LINRX_ITHRESH_I = 2) : Maximum number of residual computations allowed for refinement. Default: 10 Aggressive: Set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in err_bnds_norm and err_bnds_comp may no longer be trustworthy.

PARAMS(LA_LINRX_CWISE_I = 3) : Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Default: 1.0 (attempt componentwise convergence)

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (5*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. The solution to every right-hand side is guaranteed. < 0: If INFO = -i, the i-th argument had an illegal value > 0 and <= N: U(INFO, INFO) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+J: The solution corresponding to the Jth right-hand side is not guaranteed. The

solutions corresponding to other right- hand sides K with K > J may not be guaranteed as well, but only the first such right-hand side is reported. If a small componentwise error is not requested (PARAMS(3) = 0.0) then the Jth right-hand side is the first with a normwise error bound that is not guaranteed (the smallest J such that ERR_BNDS_NORM(J,1) = 0.0). By default (PARAMS(3) = 1.0) the Jth right-hand side is the first with either a normwise or componentwise error bound that is not guaranteed (the smallest J such that either ERR_BNDS_NORM(J,1) = 0.0 or ERR_BNDS_COMP(J,1) = 0.0). See the definition of ERR_BNDS_NORM(:,1) and ERR_BNDS_COMP(:,1). To get information about all of the right-hand sides check `ERR_BNDS_NORM` or `ERR_BNDS_COMP`.

Related Information

For this routine in other precisions, please see *zhesvxx*. It also exists with a native C interface as *LAPACKE_chesvxx*.

4.6.18 chetrs

`chetrs` solves a system of linear equations A*X = B with a complex Hermitian matrix A using the factorization A = U*D*U^H or A = L*D*L^H computed by CHETRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine chetrs(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void chetrs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const armpl_int_t *ipiv, armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = '`U`': Upper triangular, form is `A` = U*D*U^H ; = '`L`': Lower triangular, form is `A` = L*D*L^H .

N Input parameter.

`N` is `INTEGER`

The order of the matrix `A`. `N` >= 0.

NRHS Input parameter.

`NRHS` is `INTEGER`

The number of right hand sides, i.e., the number of columns of the matrix `B`. `NRHS` >= 0.

A Input parameter.

`A` is `COMPLEX`

`A` is an array, dimension (`LDA`, `N`). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CHETRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CHETRF.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zhetrs*. It also exists with a native C interface as *LAPACKE_chetrs*.

4.6.19 chetrs2

chetrs2 solves a system of linear equations A*X = B with a complex Hermitian matrix A using the factorization A = U*D*U^H or A = L*D*L^H computed by CHETRF and converted by CSYCONV.

Syntax

Fortran specification:

```
use armpl_library

subroutine chetrs2(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void chetrs2_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
              const armpl_singlecomplex_t *a, const armpl_int_t *lda,
              const armpl_int_t *ipiv, armpl_singlecomplex_t *b,
              const armpl_int_t *ldb, armpl_singlecomplex_t *work,
              armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^H ; = 'L': Lower triangular, form is A = L*D*L^H .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CHETRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CHETRF.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zhetrs2*. It also exists with a native C interface as *LAPACKE_chetrs2*.

4.6.20 chetrs_3

CHETRS_3 solves a system of linear equations A * X = B with a complex Hermitian matrix A using the factorization computed by CHETRF_RK or CHETRF_BK:

```
A = P*U*D*(U**H)*(P**T) or A = P*L*D*(L**H)*(P**T),
```

where U (or L) is unit upper (or lower) triangular matrix, U^H (or L^H) is the conjugate of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is Hermitian and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This algorithm is using Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine chetrs_3(UPLO, N, NRHS, A, LDA, E, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void chetrs_3_(const char *uplo, const armpl_int_t *n,
               const armpl_int_t *nrhs, const armpl_singlecomplex_t *a,
               const armpl_int_t *lda, const armpl_singlecomplex_t *e,
               const armpl_int_t *ipiv, armpl_singlecomplex_t *b,
               const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix: = 'U': Upper triangular, form is A = P*U*D*(U^H)*(P^T); = 'L': Lower triangular, form is A = P*L*D*(L^H)*(P^T).

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). Diagonal of the block diagonal matrix D and factors U or L as computed by CHETRF_RK and CHETRF_BK: a) ONLY diagonal elements of the Hermitian block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D should be provided on entry in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Input parameter.

E is COMPLEX

E is an array, dimension (N). On entry, contains the superdiagonal (or subdiagonal) elements of the Hermitian block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i),i=2:N, E(1) not referenced; If UPLO = 'L': E(i) = D(i+1,i),i=1:N-1, E(N) not referenced.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is not referenced in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CHETRF_RK or CHETRF_BK.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zhetrs_3*. It also exists with a native C interface as *LAPACKE_chetrs_3*.

4.6.21 chetrs_aa

CHETRS_AA solves a system of linear equations A*X = B with a complex hermitian matrix A using the factorization A = U*T*U^H or A = L*T*L^H computed by CHETRF_AA.

Syntax

Fortran specification:

```
use armpl_library

subroutine chetrs_aa(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void chetrs_aa_(const char *uplo, const armpl_int_t *n,
                const armpl_int_t *nrhs, const armpl_singlecomplex_t *a,
                const armpl_int_t *lda, const armpl_int_t *ipiv,
                armpl_singlecomplex_t *b, const armpl_int_t *ldb,
                const armpl_singlecomplex_t *work, const armpl_int_t *lwork,
                armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*T*U^H ; = 'L': Lower triangular, form is A = L*T*L^H .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). Details of factors computed by CHETRF_AA.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges as computed by CHETRF_AA.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Input parameter.

WORK is DOUBLE array, dimension (MAX(1, LWORK))

LWORK Input parameter.

LWORK is INTEGER, LWORK >= MAX(1,3*N-2).

param[out] INFO verbatim INFO is INTEGER = 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zhetrs_aa*. It also exists with a native C interface as *LAPACKE_chetrs_aa*.

4.6.22 chetrs_rook

CHETRS_ROOK solves a system of linear equations A*X = B with a complex Hermitian matrix A using the factorization A = U*D*U^H or A = L*D*L^H computed by CHETRF_ROOK.

Syntax

Fortran specification:

```
use armpl_library

subroutine chetrs_rook(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void chetrs_rook_(const char *uplo, const armpl_int_t *n,
                  const armpl_int_t *nrhs, const armpl_singlecomplex_t *a,
                  const armpl_int_t *lda, const armpl_int_t *ipiv,
                  armpl_singlecomplex_t *b, const armpl_int_t *ldb,
                  armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^H ; = 'L': Lower triangular, form is A = L*D*L^H .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CHETRF_ROOK.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CHETRF_ROOK.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zhetrs_rook*. It also exists with a native C interface as *LAPACKE_chetrs_rook*.

4.6.23 chpsv

chpsv computes the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N Hermitian matrix stored in packed format and X and B are N-by-NRHS matrices.

The diagonal pivoting method is used to factor A as

```
 A = U * D * U**H,  if UPLO = 'U', or
A = L * D * L**H,  if UPLO = 'L',
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, D is Hermitian and block diagonal with 1-by-1 and 2-by-2 diagonal blocks. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine chpsv(UPLO, N, NRHS, AP, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void chpsv_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
            armpl_singlecomplex_t *ap, armpl_int_t *ipiv,
            armpl_singlecomplex_t *b, const armpl_int_t *ldb,
            armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AP Input and output parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the Hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n. See below for further details.

On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^H or A = L*D*L^H as computed by CHPTRF, stored as a packed triangular matrix in the same storage format as A.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D, as determined by CHPTRF. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged, and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, so the solution could not be computed.

Related Information

For this routine in other precisions, please see *zhpsv*. It also exists with a native C interface as *LAPACKE_chpsv*.

4.6.24 chpsvx

chpsvx uses the diagonal pivoting factorization A = U*D*U^H or A = L*D*L^H to compute the solution to a complex system of linear equations A * X = B, where A is an N-by-N Hermitian matrix stored in packed format and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed:
```

1. If FACT = 'N', the diagonal pivoting method is used to factor A as

```
    A = U * D * U**H,  if UPLO = 'U', or
   A = L * D * L**H,  if UPLO = 'L',
where U (or L) is a product of permutation and unit upper (lower)
triangular matrices and D is Hermitian and block diagonal with
1-by-1 and 2-by-2 diagonal blocks.
```

2. If some D(i,i)=0, so that D is exactly singular, then the routine

```
 returns with INFO = i. Otherwise, the factored form of A is used
to estimate the condition number of the matrix A.  If the
reciprocal of the condition number is less than machine precision,
INFO = N+1 is returned as a warning, but the routine still goes on
to solve for X and compute error bounds as described below.
```

3. The system of equations is solved for X using the factored form

```
of A.
```

4. Iterative refinement is applied to improve the computed solution

```
 matrix and calculate error bounds and backward error estimates
for it.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine chpsvx(FACT, UPLO, N, NRHS, AP, AFP, IPIV, B, LDB, X, LDX, RCOND,
                  FERR, BERR, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void chpsvx_(const char *fact, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *nrhs, const armpl_singlecomplex_t *ap,
             armpl_singlecomplex_t *afp, armpl_int_t *ipiv,
             const armpl_singlecomplex_t *b, const armpl_int_t *ldb,
             armpl_singlecomplex_t *x, const armpl_int_t *ldx, float *rcond,
             float *ferr, float *berr, armpl_singlecomplex_t *work,
             float *rwork, armpl_int_t *info, ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of A has been supplied on entry. = 'F': On entry, AFP and IPIV contain the factored form of A. AFP and IPIV will not be modified. = 'N': The matrix A will be copied to AFP and factored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AP Input parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). The upper or lower triangle of the Hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n. See below for further details.

AFP Input and output parameter.

AFP is COMPLEX

AFP is an array, dimension (N*(N+1)/2). If FACT = 'F', then AFP is an input argument and on entry contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^H or A = L*D*L^H as computed by CHPTRF, stored as a packed triangular matrix in the same storage format as A.

If FACT = 'N', then AFP is an output argument and on exit contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^H or A = L*D*L^H as computed by CHPTRF, stored as a packed triangular matrix in the same storage format as A.

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains details of the interchanges and the block structure of D, as determined by CHPTRF. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

If FACT = 'N', then IPIV is an output argument and on exit contains details of the interchanges and the block structure of D, as determined by CHPTRF.

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). The N-by-NRHS right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is COMPLEX

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

The estimate of the reciprocal condition number of the matrix A. If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: D(i,i) is exactly zero. The factorization has been completed but the factor D is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+1: D is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *zhpsvx*. It also exists with a native C interface as *LAPACKE_chpsvx*.

4.6.25 cpbsv

cpbsv computes the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N Hermitian positive definite band matrix and X and B are N-by-NRHS matrices.

The Cholesky decomposition is used to factor A as

```
A = U**H * U,  if UPLO = 'U', or
A = L * L**H,  if UPLO = 'L',
```

where U is an upper triangular band matrix, and L is a lower triangular band matrix, with the same number of superdiagonals or subdiagonals as A. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine cpbsv(UPLO, N, KD, NRHS, AB, LDAB, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void cpbsv_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
            const armpl_int_t *nrhs, armpl_singlecomplex_t *ab,
            const armpl_int_t *ldab, armpl_singlecomplex_t *b,
            const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AB Input and output parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the Hermitian band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(KD+1+i-j,j) = A(i,j) for max(1,j-KD)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(N,j+KD). See below for further details.

On exit, if INFO = 0, the triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H of the band matrix A, in the same storage format as A.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

Related Information

For this routine in other precisions, please see *dpbsv*, *spbsv* and *zpbsv*. It also exists with a native C interface as *LAPACKE_cpbsv*.

4.6.26 cpbsvx

cpbsvx uses the Cholesky factorization A = U^H *U or A = L*L^H to compute the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N Hermitian positive definite band matrix and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed:
```

1. If FACT = 'E', real scaling factors are computed to equilibrate

```
 the system:
   diag(S) * A * diag(S) * inv(diag(S)) * X = diag(S) * B
Whether or not the system will be equilibrated depends on the
scaling of the matrix A, but if equilibration is used, A is
overwritten by diag(S)*A*diag(S) and B by diag(S)*B.
```

2. If FACT = 'N' or 'E', the Cholesky decomposition is used to

```
 factor the matrix A (after equilibration if FACT = 'E') as
   A = U**H * U,  if UPLO = 'U', or
   A = L * L**H,  if UPLO = 'L',
where U is an upper triangular band matrix, and L is a lower
triangular band matrix.
```

3. If the leading i-by-i principal minor is not positive definite,

```
 then the routine returns with INFO = i. Otherwise, the factored
form of A is used to estimate the condition number of the matrix
A.  If the reciprocal of the condition number is less than machine
precision, INFO = N+1 is returned as a warning, but the routine
still goes on to solve for X and compute error bounds as
described below.
```

4. The system of equations is solved for X using the factored form

```
of A.
```

5. Iterative refinement is applied to improve the computed solution

```
 matrix and calculate error bounds and backward error estimates
for it.
```

6. If equilibration was used, the matrix X is premultiplied by

```
 diag(S) so that it solves the original system before
equilibration.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine cpbsvx(FACT, UPLO, N, KD, NRHS, AB, LDAB, AFB, LDAFB, EQUED, S, B,
                  LDB, X, LDX, RCOND, FERR, BERR, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cpbsvx_(const char *fact, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *kd, const armpl_int_t *nrhs,
             armpl_singlecomplex_t *ab, const armpl_int_t *ldab,
             armpl_singlecomplex_t *afb, const armpl_int_t *ldafb,
             char *equed, float *s, armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, armpl_singlecomplex_t *x,
             const armpl_int_t *ldx, float *rcond, float *ferr, float *berr,
             armpl_singlecomplex_t *work, float *rwork, armpl_int_t *info,
             ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AFB contains the factored form of A. If EQUED = 'Y', the matrix A has been equilibrated with scaling factors given by S. AB and AFB will not be modified. = 'N': The matrix A will be copied to AFB and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AFB and factored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right-hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AB Input and output parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the Hermitian band matrix A, stored in the first KD+1 rows of the array, except if FACT = 'F' and EQUED = 'Y', then A must contain the equilibrated matrix diag(S)*A*diag(S). The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(KD+1+i-j,j) = A(i,j) for max(1,j-KD)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(N,j+KD). See below for further details.

On exit, if FACT = 'E' and EQUED = 'Y', A is overwritten by diag(S)*A*diag(S).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array A. LDAB >= KD+1.

AFB Input and output parameter.

AFB is COMPLEX

AFB is an array, dimension (LDAFB, N). If FACT = 'F', then AFB is an input argument and on entry contains the triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H of the band matrix A, in the same storage format as A (see AB). If EQUED = 'Y', then AFB is the factored form of the equilibrated matrix A.

If FACT = 'N', then AFB is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H .

If FACT = 'E', then AFB is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H of the equilibrated matrix A (see the description of A for the form of the equilibrated matrix).

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AFB. LDAFB >= KD+1.

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'Y': Equilibration was done, i.e., A has been replaced by diag(S) * A * diag(S). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

S Input and output parameter.

S is REAL

S is an array, dimension (N). The scale factors for A; not accessed if EQUED = 'N'. S is an input argument if FACT = 'F'; otherwise, S is an output argument. If FACT = 'F' and EQUED = 'Y', each element of S must be positive.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if EQUED = 'N', B is not modified; if EQUED = 'Y', B is overwritten by diag(S) * B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is COMPLEX

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X to the original system of equations. Note that if EQUED = 'Y', A and B are modified on exit, and the solution to the equilibrated system is inv(diag(S))*X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

The estimate of the reciprocal condition number of the matrix A after equilibration (if done). If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed. RCOND = 0 is returned. = N+1: U is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *dpbsvx*, *spbsvx* and *zpbsvx*. It also exists with a native C interface as *LAPACKE_cpbsvx*.

4.6.27 cpbtrs

`cpbtrs` solves a system of linear equations A*X = B with a Hermitian positive definite band matrix A using the Cholesky factorization A = U^H *U or A = L*L^H computed by CPBTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine cpbtrs(UPLO, N, KD, NRHS, AB, LDAB, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void cpbtrs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
             const armpl_int_t *nrhs, const armpl_singlecomplex_t *ab,
             const armpl_int_t *ldab, armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

= '`U`': Upper triangular factor stored in AB; = '`L`': Lower triangular factor stored in `AB`.

N Input parameter.

`N` is `INTEGER`

The order of the matrix A. `N` >= 0.

KD Input parameter.

`KD` is `INTEGER`

The number of superdiagonals of the matrix A if `UPLO` = '`U`', or the number of subdiagonals if `UPLO` = '`L`'. `KD` >= 0.

NRHS Input parameter.

`NRHS` is `INTEGER`

The number of right hand sides, i.e., the number of columns of the matrix `B`. `NRHS` >= 0.

AB Input parameter.

`AB` is `COMPLEX`

`AB` is an array, dimension (`LDAB`, `N`). The triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H of the band matrix A, stored in the first KD+1 rows of the array. The j-th column of U or L is stored in the j-th column of the array `AB` as follows: if `UPLO` ='`U`', AB(kd+1+i-j,j) = U(i,j) for max(1,j-kd)<=i<=j; if `UPLO` ='`L`', AB(1+i-j,j) = L(i,j) for j<=i<=min(n,j+kd).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dpbtrs*, *spbtrs* and *zpbtrs*. It also exists with a native C interface as *LAPACKE_cpbtrs*.

4.6.28 cpftrs

cpftrs solves a system of linear equations A*X = B with a Hermitian positive definite matrix A using the Cholesky factorization A = U^H *U or A = L*L^H computed by CPFTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine cpftrs(TRANSR, UPLO, N, NRHS, A, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void cpftrs_(const char *transr, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *nrhs, const armpl_singlecomplex_t *a,
             armpl_singlecomplex_t *b, const armpl_int_t *ldb,
             armpl_int_t *info, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': The Normal TRANSR of RFP A is stored; = 'C': The Conjugate-transpose TRANSR of RFP A is stored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of RFP A is stored; = 'L': Lower triangle of RFP A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (N*(N+1)/2);. The triangular factor U or L from the Cholesky factorization of RFP A = U^H *U or RFP A = L*L^H , as computed by CPFTRF. See note below for more details about RFP A.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dpftrs*, *spftrs* and *zpftrs*. It also exists with a native C interface as *LAPACKE_cpftrs*.

4.6.29 cposv

cposv computes the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N Hermitian positive definite matrix and X and B are N-by-NRHS matrices.

The Cholesky decomposition is used to factor A as

```
A = U**H* U,  if UPLO = 'U', or
A = L * L**H,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is a lower triangular matrix. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine cposv(UPLO, N, NRHS, A, LDA, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void cposv_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
            armpl_singlecomplex_t *a, const armpl_int_t *lda,
            armpl_singlecomplex_t *b, const armpl_int_t *ldb,
            armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H .

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

Related Information

For this routine in other precisions, please see *dposv*, *sposv* and *zposv*. It also exists with a native C interface as *LAPACKE_cposv*.

4.6.30 cposvx

cposvx uses the Cholesky factorization A = U^H *U or A = L*L^H to compute the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N Hermitian positive definite matrix and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed:
```

1. If FACT = 'E', real scaling factors are computed to equilibrate

```
 the system:
   diag(S) * A * diag(S) * inv(diag(S)) * X = diag(S) * B
Whether or not the system will be equilibrated depends on the
scaling of the matrix A, but if equilibration is used, A is
overwritten by diag(S)*A*diag(S) and B by diag(S)*B.
```

2. If FACT = 'N' or 'E', the Cholesky decomposition is used to

```
 factor the matrix A (after equilibration if FACT = 'E') as
   A = U**H* U,  if UPLO = 'U', or
   A = L * L**H,  if UPLO = 'L',
where U is an upper triangular matrix and L is a lower triangular
matrix.
```

3. If the leading i-by-i principal minor is not positive definite,

```
 then the routine returns with INFO = i. Otherwise, the factored
form of A is used to estimate the condition number of the matrix
A.  If the reciprocal of the condition number is less than machine
precision, INFO = N+1 is returned as a warning, but the routine
still goes on to solve for X and compute error bounds as
described below.
```

4. The system of equations is solved for X using the factored form

```
of A.
```

5. Iterative refinement is applied to improve the computed solution

```
 matrix and calculate error bounds and backward error estimates
for it.
```

6. If equilibration was used, the matrix X is premultiplied by

```
 diag(S) so that it solves the original system before
equilibration.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine cposvx(FACT, UPLO, N, NRHS, A, LDA, AF, LDAF, EQUED, S, B, LDB, X,
                  LDX, RCOND, FERR, BERR, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cposvx_(const char *fact, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *nrhs, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_singlecomplex_t *af,
             const armpl_int_t *ldaf, char *equed, float *s,
             armpl_singlecomplex_t *b, const armpl_int_t *ldb,
             armpl_singlecomplex_t *x, const armpl_int_t *ldx, float *rcond,
             float *ferr, float *berr, armpl_singlecomplex_t *work,
             float *rwork, armpl_int_t *info, ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AF contains the factored form of A. If EQUED = 'Y', the matrix A has been equilibrated with scaling factors given by S. A and AF will not be modified. = 'N': The matrix A will be copied to AF and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AF and factored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A, except if FACT = 'F' and EQUED = 'Y', then A must contain the equilibrated matrix diag(S)*A*diag(S). If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. A is not modified if FACT = 'F' or 'N', or if FACT = 'E' and EQUED = 'N' on exit.

On exit, if FACT = 'E' and EQUED = 'Y', A is overwritten by diag(S)*A*diag(S).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input and output parameter.

AF is COMPLEX

AF is an array, dimension (LDAF, N). If FACT = 'F', then AF is an input argument and on entry contains the triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H , in the same storage format as A. If EQUED .ne. 'N', then AF is the factored form of the equilibrated matrix diag(S)*A*diag(S).

If FACT = 'N', then AF is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H of the original matrix A.

If FACT = 'E', then AF is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H of the equilibrated matrix A (see the description of A for the form of the equilibrated matrix).

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'Y': Equilibration was done, i.e., A has been replaced by diag(S) * A * diag(S). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

S Input and output parameter.

S is REAL

S is an array, dimension (N). The scale factors for A; not accessed if EQUED = 'N'. S is an input argument if FACT = 'F'; otherwise, S is an output argument. If FACT = 'F' and EQUED = 'Y', each element of S must be positive.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS righthand side matrix B. On exit, if EQUED = 'N', B is not modified; if EQUED = 'Y', B is overwritten by diag(S) * B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is COMPLEX

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X to the original system of equations. Note that if EQUED = 'Y', A and B are modified on exit, and the solution to the equilibrated system is inv(diag(S))*X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

The estimate of the reciprocal condition number of the matrix A after equilibration (if done). If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed. RCOND = 0 is returned. = N+1: U is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *dposvx*, *sposvx* and *zposvx*. It also exists with a native C interface as *LAPACKE_cposvx*.

4.6.31 cposvxx

```
CPOSVXX uses the Cholesky factorization A = U**T*U or A = L*L**T
   to compute the solution to a complex system of linear equations
   A * X = B, where A is an N-by-N symmetric positive definite matrix
   and X and B are N-by-NRHS matrices.

   If requested, both normwise and maximum componentwise error bounds
   are returned. CPOSVXX will return a solution with a tiny
   guaranteed error (O(eps) where eps is the working machine
   precision) unless the matrix is very ill-conditioned, in which
   case a warning is returned. Relevant condition numbers also are
   calculated and returned.

   CPOSVXX accepts user-provided factorizations and equilibration
   factors; see the definitions of the FACT and EQUED options.
   Solving with refinement and using a factorization from a previous
   CPOSVXX call will also produce a solution with either O(eps)
   errors or warnings, but we cannot make that claim for general
```

(continues on next page)

(continued from previous page)

```
   user-provided factorizations and equilibration factors if they
   differ from what CPOSVXX would itself produce.

                                                                    The␣
↪following steps are performed:

   1. If FACT = 'E', real scaling factors are computed to equilibrate
   the system:

     diag(S)*A*diag(S)      *inv(diag(S))*X = diag(S)*B

   Whether or not the system will be equilibrated depends on the
   scaling of the matrix A, but if equilibration is used, A is
   overwritten by diag(S)*A*diag(S) and B by diag(S)*B.

   2. If FACT = 'N' or 'E', the Cholesky decomposition is used to
   factor the matrix A (after equilibration if FACT = 'E') as
      A = U**T* U,  if UPLO = 'U', or
      A = L * L**T,  if UPLO = 'L',
   where U is an upper triangular matrix and L is a lower triangular
   matrix.

   3. If the leading i-by-i principal minor is not positive definite,
   then the routine returns with INFO = i. Otherwise, the factored
   form of A is used to estimate the condition number of the matrix
   A (see argument RCOND).  If the reciprocal of the condition number
   is less than machine precision, the routine still goes on to solve
   for X and compute error bounds as described below.

   4. The system of equations is solved for X using the factored form
   of A.

   5. By default (unless PARAMS(LA_LINRX_ITREF_I) is set to zero),
   the routine will use iterative refinement to try to get a small
   error and error bounds.  Refinement calculates the residual to at
   least twice the working precision.

   6. If equilibration was used, the matrix X is premultiplied by
   diag(S) so that it solves the original system before
   equilibration.                          Some optional parameters are bundled␣
↪in the PARAMS array.  These
    settings determine how refinement is performed, but often the
    defaults are acceptable.  If the defaults are acceptable, users
    can pass NPARAMS = 0 which prevents the source code from accessing
    the PARAMS argument.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine cposvxx(FACT, UPLO, N, NRHS, A, LDA, AF, LDAF, EQUED, S, B, LDB, X,
                   LDX, RCOND, RPVGRW, BERR, N_ERR_BNDS, ERR_BNDS_NORM,
                   ERR_BNDS_COMP, NPARAMS, PARAMS, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"
```

(continues on next page)

(continued from previous page)

```
void cposvxx_(const char *fact, const char *uplo, const armpl_int_t *n,
              const armpl_int_t *nrhs, armpl_singlecomplex_t *a,
              const armpl_int_t *lda, armpl_singlecomplex_t *af,
              const armpl_int_t *ldaf, char *equed, float *s,
              armpl_singlecomplex_t *b, const armpl_int_t *ldb,
              armpl_singlecomplex_t *x, const armpl_int_t *ldx, float *rcond,
              float *rpvgrw, float *berr, const armpl_int_t *n_err_bnds,
              float *err_bnds_norm, float *err_bnds_comp,
              const armpl_int_t *nparams, float *params,
              armpl_singlecomplex_t *work, float *rwork, armpl_int_t *info,
              ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AF contains the factored form of A. If EQUED is not 'N', the matrix A has been equilibrated with scaling factors given by S. A and AF are not modified. = 'N': The matrix A will be copied to AF and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AF and factored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the symmetric matrix A, except if FACT = 'F' and EQUED = 'Y', then A must contain the equilibrated matrix diag(S)*A*diag(S). If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. A is not modified if FACT = 'F' or 'N', or if FACT = 'E' and EQUED = 'N' on exit.

On exit, if FACT = 'E' and EQUED = 'Y', A is overwritten by diag(S)*A*diag(S).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input and output parameter.

AF is COMPLEX

AF is an array, dimension (LDAF, N). If FACT = 'F', then AF is an input argument and on entry contains the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , in the same storage format as A. If EQUED .ne. 'N', then AF is the factored form of the equilibrated matrix diag(S)*A*diag(S).

If FACT = 'N', then AF is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T of the original matrix A.

If FACT = 'E', then AF is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T of the equilibrated matrix A (see the description of A for the form of the equilibrated matrix).

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'Y': Both row and column equilibration, i.e., A has been replaced by diag(S) * A * diag(S). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

S Input and output parameter.

S is REAL

S is an array, dimension (N). The row scale factors for A. If EQUED = 'Y', A is multiplied on the left and right by diag(S). S is an input argument if FACT = 'F'; otherwise, S is an output argument. If FACT = 'F' and EQUED = 'Y', each element of S must be positive. If S is output, each element of S is a power of the radix. If S is input, each element of S should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if EQUED = 'N', B is not modified; if EQUED = 'Y', B is overwritten by diag(S)*B;

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is COMPLEX

X is an array, dimension (LDX, NRHS). If INFO = 0, the N-by-NRHS solution matrix X to the original system of equations. Note that A and B are modified on exit if EQUED .ne. 'N', and the solution to the equilibrated system is inv(diag(S))*X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the

matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

RPVGRW Output parameter.

RPVGRW is REAL

Reciprocal pivot growth. On exit, this contains the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If this is much less than 1, then the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, estimated condition numbers, and error bounds could be unreliable. If factorization fails with 0<INFO<=N, then this contains the reciprocal pivot growth factor for the leading INFO columns of A.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). Componentwise relative backward error. This is the componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

N_ERR_BNDS Input parameter.

N_ERR_BNDS is INTEGER

Number of error bounds to return for each right hand side and each type (normwise or componentwise). See ERR_BNDS_NORM and ERR_BNDS_COMP below.

ERR_BNDS_NORM Output parameter.

ERR_BNDS_NORM is REAL

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Output parameter.

ERR_BNDS_COMP is REAL

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:, N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

NPARAMS Input parameter.

NPARAMS is INTEGER

Specifies the number of parameters set in PARAMS. If .LE. 0, the PARAMS array is never referenced and default values are used.

PARAMS Input and output parameter.

PARAMS is REAL

PARAMS is an array, dimension NPARAMS. Specifies algorithm parameters. If an entry is .LT. 0.0, then that entry will be filled with default value used for that parameter. Only positions up to NPARAMS are accessed; defaults are used for higher-numbered parameters.

PARAMS(LA_LINRX_ITREF_I = 1) : Whether to perform iterative refinement or not. Default: 1.0 = 0.0 : No refinement is performed, and no error bounds are computed. = 1.0 : Use the double-precision refinement algorithm, possibly with doubled-single computations if the compilation environment does not support DOUBLE PRECISION. (other values are reserved for future use)

PARAMS(LA_LINRX_ITHRESH_I = 2) : Maximum number of residual computations allowed for refinement. Default: 10 Aggressive: Set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in err_bnds_norm and err_bnds_comp may no longer be trustworthy.

PARAMS(LA_LINRX_CWISE_I = 3) : Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Default: 1.0 (attempt componentwise convergence)

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. The solution to every right-hand side is guaranteed. < 0: If INFO = -i, the i-th argument had an illegal value > 0 and <= N: U(INFO, INFO) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+J: The solution corresponding to the Jth right-hand side is not guaranteed. The solutions corresponding to other right- hand sides K with K > J may not be guaranteed as well, but only the first such right-hand side is reported. If a small componentwise error is not requested (PARAMS(3) = 0.0) then the Jth right-hand side is the first with a normwise error bound that is not guaranteed (the smallest J such that ERR_BNDS_NORM(J,1) = 0.0). By default (PARAMS(3) = 1.0) the Jth right-hand side is the first with either a normwise or componentwise error bound that is not guaranteed (the smallest J such that either ERR_BNDS_NORM(J,1) = 0.0 or ERR_BNDS_COMP(J,1) = 0.0). See the definition of ERR_BNDS_NORM(:,1) and ERR_BNDS_COMP(:,1). To get information about all of the right-hand sides check ERR_BNDS_NORM or ERR_BNDS_COMP.

Related Information

For this routine in other precisions, please see *dposvxx*, *sposvxx* and *zposvxx*. It also exists with a native C interface as *LAPACKE_cposvxx*.

4.6.32 cpotrs

cpotrs solves a system of linear equations A*X = B with a Hermitian positive definite matrix A using the Cholesky factorization $A = U^H * U$ or $A = L * L^H$ computed by CPOTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine cpotrs(UPLO, N, NRHS, A, LDA, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void cpotrs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda,
             armpl_singlecomplex_t *b, const armpl_int_t *ldb,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H , as computed by CPOTRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dpotrs*, *spotrs* and *zpotrs*. It also exists with a native C interface as *LAPACKE_cpotrs*.

4.6.33 cppsv

cppsv computes the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N Hermitian positive definite matrix stored in packed format and X and B are N-by-NRHS matrices.

The Cholesky decomposition is used to factor A as

```
A = U**H * U,  if UPLO = 'U', or
A = L * L**H,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is a lower triangular matrix. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine cppsv(UPLO, N, NRHS, AP, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void cppsv_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
            armpl_singlecomplex_t *ap, armpl_singlecomplex_t *b,
            const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AP Input and output parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the Hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n. See below for further details.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H , in the same storage format as A.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

Related Information

For this routine in other precisions, please see *dppsv*, *sppsv* and *zppsv*. It also exists with a native C interface as *LAPACKE_cppsv*.

4.6.34 cppsvx

cppsvx uses the Cholesky factorization A = U^H *U or A = L*L^H to compute the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N Hermitian positive definite matrix stored in packed format and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed:
```

1. If FACT = 'E', real scaling factors are computed to equilibrate

```
 the system:
   diag(S) * A * diag(S) * inv(diag(S)) * X = diag(S) * B
Whether or not the system will be equilibrated depends on the
scaling of the matrix A, but if equilibration is used, A is
overwritten by diag(S)*A*diag(S) and B by diag(S)*B.
```

2. If FACT = 'N' or 'E', the Cholesky decomposition is used to

```
 factor the matrix A (after equilibration if FACT = 'E') as
   A = U**H * U ,  if UPLO = 'U', or
   A = L * L**H,  if UPLO = 'L',
where U is an upper triangular matrix, L is a lower triangular
matrix, and **H indicates conjugate transpose.
```

3. If the leading i-by-i principal minor is not positive definite,

```
 then the routine returns with INFO = i. Otherwise, the factored
form of A is used to estimate the condition number of the matrix
A.  If the reciprocal of the condition number is less than machine
precision, INFO = N+1 is returned as a warning, but the routine
still goes on to solve for X and compute error bounds as
described below.
```

4. The system of equations is solved for X using the factored form

```
of A.
```

5. Iterative refinement is applied to improve the computed solution

```
 matrix and calculate error bounds and backward error estimates
for it.
```

6. If equilibration was used, the matrix X is premultiplied by

```
 diag(S) so that it solves the original system before
equilibration.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine cppsvx(FACT, UPLO, N, NRHS, AP, AFP, EQUED, S, B, LDB, X, LDX,
                  RCOND, FERR, BERR, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cppsvx_(const char *fact, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *nrhs, armpl_singlecomplex_t *ap,
             armpl_singlecomplex_t *afp, char *equed, float *s,
             armpl_singlecomplex_t *b, const armpl_int_t *ldb,
             armpl_singlecomplex_t *x, const armpl_int_t *ldx, float *rcond,
             float *ferr, float *berr, armpl_singlecomplex_t *work,
             float *rwork, armpl_int_t *info, ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AFP contains the factored form of A. If EQUED = 'Y', the matrix A has been equilibrated with scaling factors given by S. AP and AFP will not be modified. = 'N': The matrix A will be copied to AFP and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AFP and factored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AP Input and output parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the Hermitian matrix A, packed columnwise in a linear array, except if FACT = 'F' and EQUED = 'Y', then A must contain the equilibrated matrix diag(S)*A*diag(S). The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n. See below for further details. A is not modified if FACT = 'F' or 'N', or if FACT = 'E' and EQUED = 'N' on exit.

On exit, if FACT = 'E' and EQUED = 'Y', A is overwritten by diag(S)*A*diag(S).

AFP Input and output parameter.

AFP is COMPLEX

AFP is an array, dimension (N*(N+1)/2). If FACT = 'F', then AFP is an input argument and on entry contains the triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H , in the same storage format as A. If EQUED .ne. 'N', then AFP is the factored form of the equilibrated matrix A.

If FACT = 'N', then AFP is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization A = U^H * U or A = L * L^H of the original matrix A.

If FACT = 'E', then AFP is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H of the equilibrated matrix A (see the description of AP for the form of the equilibrated matrix).

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'Y': Equilibration was done, i.e., A has been replaced by diag(S) * A * diag(S). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

S Input and output parameter.

S is REAL

S is an array, dimension (N). The scale factors for A; not accessed if EQUED = 'N'. S is an input argument if FACT = 'F'; otherwise, S is an output argument. If FACT = 'F' and EQUED = 'Y', each element of S must be positive.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if EQUED = 'N', B is not modified; if EQUED = 'Y', B is overwritten by diag(S) * B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is COMPLEX

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X to the original system of equations. Note that if EQUED = 'Y', A and B are modified on exit, and the solution to the equilibrated system is inv(diag(S))*X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

The estimate of the reciprocal condition number of the matrix A after equilibration (if done). If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed. RCOND = 0 is returned. = N+1: U is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *dppsvx*, *sppsvx* and *zppsvx*. It also exists with a native C interface as *LAPACKE_cppsvx*.

4.6.35 cpptrs

cpptrs solves a system of linear equations A*X = B with a Hermitian positive definite matrix A in packed storage using the Cholesky factorization A = U^H *U or A = L*L^H computed by CPPTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine cpptrs(UPLO, N, NRHS, AP, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void cpptrs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_singlecomplex_t *ap, armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AP Input parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). The triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H , packed columnwise in a linear array. The j-th column of U or L is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = U(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = L(i,j) for j<=i<=n.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dpptrs*, *spptrs* and *zpptrs*. It also exists with a native C interface as *LAPACKE_cpptrs*.

4.6.36 cptsv

cptsv computes the solution to a complex system of linear equations A*X = B, where A is an N-by-N Hermitian positive definite tridiagonal matrix, and X and B are N-by-NRHS matrices.

A is factored as A = L*D*L^H , and the factored form of A is then used to solve the system of equations.

Syntax

Fortran specification:

```
use armpl_library

subroutine cptsv(N, NRHS, D, E, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void cptsv_(const armpl_int_t *n, const armpl_int_t *nrhs, float *d,
            armpl_singlecomplex_t *e, armpl_singlecomplex_t *b,
            const armpl_int_t *ldb, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry, the n diagonal elements of the tridiagonal matrix A. On exit, the n diagonal elements of the diagonal matrix D from the factorization A = L*D*L^H .

E Input and output parameter.

E is COMPLEX

E is an array, dimension (N-1). On entry, the (n-1) subdiagonal elements of the tridiagonal matrix A. On exit, the (n-1) subdiagonal elements of the unit bidiagonal factor L from the L*D*L^H factorization of A. E can also be regarded as the superdiagonal of the unit bidiagonal factor U from the U^H *D*U factorization of A.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i is not positive definite, and the solution has not been computed. The factorization has not been completed unless i = N.

Related Information

For this routine in other precisions, please see *dptsv*, *sptsv* and *zptsv*. It also exists with a native C interface as *LAPACKE_cptsv*.

4.6.37 cptsvx

cptsvx uses the factorization A = L*D*L^H to compute the solution to a complex system of linear equations A*X = B, where A is an N-by-N Hermitian positive definite tridiagonal matrix and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed:
```

1. If FACT = 'N', the matrix A is factored as A = L*D*L^H , where L

```
is a unit lower bidiagonal matrix and D is diagonal.  The
factorization can also be regarded as having the form
A = U**H*D*U.
```

2. If the leading i-by-i principal minor is not positive definite,

```
then the routine returns with INFO = i. Otherwise, the factored
form of A is used to estimate the condition number of the matrix
A.  If the reciprocal of the condition number is less than machine
precision, INFO = N+1 is returned as a warning, but the routine
still goes on to solve for X and compute error bounds as
described below.
```

3. The system of equations is solved for X using the factored form

```
of A.
```

4. Iterative refinement is applied to improve the computed solution

```
matrix and calculate error bounds and backward error estimates
for it.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine cptsvx(FACT, N, NRHS, D, E, DF, EF, B, LDB, X, LDX, RCOND, FERR,
                  BERR, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cptsvx_(const char *fact, const armpl_int_t *n, const armpl_int_t *nrhs,
             const float *d, const armpl_singlecomplex_t *e, float *df,
             armpl_singlecomplex_t *ef, const armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, armpl_singlecomplex_t *x,
             const armpl_int_t *ldx, float *rcond, float *ferr, float *berr,
             armpl_singlecomplex_t *work, float *rwork, armpl_int_t *info,
             ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry. = 'F': On entry, DF and EF contain the factored form of A. D, E, DF, and EF will not be modified. = 'N': The matrix A will be copied to DF and EF and factored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

D Input parameter.

D is REAL

D is an array, dimension (N). The n diagonal elements of the tridiagonal matrix A.

E Input parameter.

E is COMPLEX

E is an array, dimension (N-1). The (n-1) subdiagonal elements of the tridiagonal matrix A.

DF Input and output parameter.

DF is REAL

DF is an array, dimension (N). If FACT = 'F', then DF is an input argument and on entry contains the n diagonal elements of the diagonal matrix D from the $L*D*L^H$ factorization of A. If FACT = 'N', then DF is an output argument and on exit contains the n diagonal elements of the diagonal matrix D from the $L*D*L^H$ factorization of A.

EF Input and output parameter.

EF is COMPLEX

EF is an array, dimension (N-1). If FACT = 'F', then EF is an input argument and on entry contains the (n-1) subdiagonal elements of the unit bidiagonal factor L from the $L*D*L^H$ factorization of A. If FACT = 'N', then EF is an output argument and on exit contains the (n-1) subdiagonal elements of the unit bidiagonal factor L from the $L*D*L^H$ factorization of A.

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). The N-by-NRHS right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is COMPLEX

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

The reciprocal condition number of the matrix A. If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j).

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed. RCOND = 0 is returned. = N+1: U is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *dptsvx*, *sptsvx* and *zptsvx*. It also exists with a native C interface as *LAPACKE_cptsvx*.

4.6.38 cpttrs

cpttrs solves a tridiagonal system of the form

```
A * X = B
```

using the factorization A = U^H *D*U or A = L*D*L^H computed by CPTTRF. D is a diagonal matrix specified in the vector D, U (or L) is a unit bidiagonal matrix whose superdiagonal (subdiagonal) is specified in the vector E, and X and B are N by NRHS matrices.

Syntax

Fortran specification:

```
use armpl_library

subroutine cpttrs(UPLO, N, NRHS, D, E, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void cpttrs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const float *d, const armpl_singlecomplex_t *e,
             armpl_singlecomplex_t *b, const armpl_int_t *ldb,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies the form of the factorization and whether the vector E is the superdiagonal of the upper bidiagonal factor U or the subdiagonal of the lower bidiagonal factor L. = 'U': A = U^H *D*U, E is the superdiagonal of U = 'L': A = L*D*L^H , E is the subdiagonal of L

N Input parameter.

N is INTEGER

The order of the tridiagonal matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

D Input parameter.

D is REAL

D is an array, dimension (N). The n diagonal elements of the diagonal matrix D from the factorization A = U^H *D*U or A = L*D*L^H .

E Input parameter.

E is COMPLEX

E is an array, dimension (N-1). If UPLO = 'U', the (n-1) superdiagonal elements of the unit bidiagonal factor U from the factorization A = U^H *D*U. If UPLO = 'L', the (n-1) subdiagonal elements of the unit bidiagonal factor L from the factorization A = L*D*L^H .

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the right hand side vectors B for the system of linear equations. On exit, the solution vectors, X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dpttrs*, *spttrs* and *zpttrs*. It also exists with a native C interface as *LAPACKE_cpttrs*.

4.6.39 cspsv

cspsv computes the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N symmetric matrix stored in packed format and X and B are N-by-NRHS matrices.

The diagonal pivoting method is used to factor A as

```
A = U * D * U**T,  if UPLO = 'U', or
A = L * D * L**T,  if UPLO = 'L',
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine cspsv(UPLO, N, NRHS, AP, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void cspsv_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
            armpl_singlecomplex_t *ap, armpl_int_t *ipiv,
            armpl_singlecomplex_t *b, const armpl_int_t *ldb,
            armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AP Input and output parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n. See below for further details.

On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by CSPTRF, stored as a packed triangular matrix in the same storage format as A.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D, as determined by CSPTRF. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged, and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, so the solution could not be computed.

Related Information

For this routine in other precisions, please see *dspsv*, *sspsv* and *zspsv*. It also exists with a native C interface as *LAPACKE_cspsv*.

4.6.40 cspsvx

cspsvx uses the diagonal pivoting factorization A = U*D*U^T or A = L*D*L^T to compute the solution to a complex system of linear equations A * X = B, where A is an N-by-N symmetric matrix stored in packed format and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed:
```

1. If FACT = 'N', the diagonal pivoting method is used to factor A as

```
   A = U * D * U**T,  if UPLO = 'U', or
   A = L * D * L**T,  if UPLO = 'L',
where U (or L) is a product of permutation and unit upper (lower)
triangular matrices and D is symmetric and block diagonal with
1-by-1 and 2-by-2 diagonal blocks.
```

2. If some D(i,i)=0, so that D is exactly singular, then the routine

```
returns with INFO = i. Otherwise, the factored form of A is used
to estimate the condition number of the matrix A.  If the
reciprocal of the condition number is less than machine precision,
INFO = N+1 is returned as a warning, but the routine still goes on
to solve for X and compute error bounds as described below.
```

3. The system of equations is solved for X using the factored form

```
of A.
```

4. Iterative refinement is applied to improve the computed solution

```
matrix and calculate error bounds and backward error estimates
for it.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine cspsvx(FACT, UPLO, N, NRHS, AP, AFP, IPIV, B, LDB, X, LDX, RCOND,
                  FERR, BERR, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cspsvx_(const char *fact, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *nrhs, const armpl_singlecomplex_t *ap,
             armpl_singlecomplex_t *afp, armpl_int_t *ipiv,
             const armpl_singlecomplex_t *b, const armpl_int_t *ldb,
             armpl_singlecomplex_t *x, const armpl_int_t *ldx, float *rcond,
             float *ferr, float *berr, armpl_singlecomplex_t *work,
             float *rwork, armpl_int_t *info, ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of A has been supplied on entry. = 'F': On entry, AFP and IPIV contain the factored form of A. AP, AFP and IPIV will not be modified. = 'N': The matrix A will be copied to AFP and factored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AP Input parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). The upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n. See below for further details.

AFP Input and output parameter.

AFP is COMPLEX

AFP is an array, dimension (N*(N+1)/2). If FACT = 'F', then AFP is an input argument and on entry contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by CSPTRF, stored as a packed triangular matrix in the same storage format as A.

If FACT = 'N', then AFP is an output argument and on exit contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by CSPTRF, stored as a packed triangular matrix in the same storage format as A.

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains details of the interchanges and the block structure of D, as determined by CSPTRF. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

If FACT = 'N', then IPIV is an output argument and on exit contains details of the interchanges and the block structure of D, as determined by CSPTRF.

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). The N-by-NRHS right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is COMPLEX

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

The estimate of the reciprocal condition number of the matrix A. If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: D(i,i) is exactly zero. The factorization has been completed but the factor D is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+1: D is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *dspsvx*, *sspsvx* and *zspsvx*. It also exists with a native C interface as *LAPACKE_cspsvx*.

4.6.41 csptrs

csptrs solves a system of linear equations A*X = B with a complex symmetric matrix A stored in packed format using the factorization A = U*D*U^T or A = L*D*L^T computed by CSPTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine csptrs(UPLO, N, NRHS, AP, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void csptrs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_singlecomplex_t *ap, const armpl_int_t *ipiv,
             armpl_singlecomplex_t *b, const armpl_int_t *ldb,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AP Input parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CSPTRF, stored as a packed triangular matrix.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CSPTRF.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dsptrs*, *ssptrs* and *zsptrs*. It also exists with a native C interface as *LAPACKE_csptrs*.

4.6.42 csysv

csysv computes the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N symmetric matrix and X and B are N-by-NRHS matrices.

The diagonal pivoting method is used to factor A as

```
A = U * D * U**T,  if UPLO = 'U', or
A = L * D * L**T,  if UPLO = 'L',
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine csysv(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void csysv_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
            armpl_singlecomplex_t *a, const armpl_int_t *lda,
            armpl_int_t *ipiv, armpl_singlecomplex_t *b,
            const armpl_int_t *ldb, armpl_singlecomplex_t *work,
            const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by CSYTRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D, as determined by CSYTRF. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged, and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= 1, and for best performance LWORK >= max(1,N*NB), where NB is the optimal blocksize for CSYTRF. for LWORK < N, TRS will be done with Level BLAS 2 for LWORK >= N, TRS will be done with Level BLAS 3

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, so the solution could not be computed.

Related Information

For this routine in other precisions, please see *dsysv*, *ssysv* and *zsysv*. It also exists with a native C interface as *LAPACKE_csysv*.

4.6.43 csysv_aa

CSYSV computes the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N symmetric matrix and X and B are N-by-NRHS matrices.

Aasen's algorithm is used to factor A as

```
A = U * T * U**T,  if UPLO = 'U', or
A = L * T * L**T,  if UPLO = 'L',
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and T is symmetric tridiagonal. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine csysv_aa(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void csysv_aa_(const char *uplo, const armpl_int_t *n,
               const armpl_int_t *nrhs, armpl_singlecomplex_t *a,
```

(continues on next page)

(continued from previous page)

```
                const armpl_int_t *lda, armpl_int_t *ipiv,
                armpl_singlecomplex_t *b, const armpl_int_t *ldb,
                armpl_singlecomplex_t *work, const armpl_int_t *lwork,
                armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the tridiagonal matrix T and the multipliers used to obtain the factor U or L from the factorization A = U*T*U^T or A = L*T*L^T as computed by CSYTRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

On exit, it contains the details of the interchanges, i.e., the row and column k of A were interchanged with the row and column IPIV(k).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= MAX(2*N, 3*N-2), and for the best performance, LWORK >= max(1,N*NB), where NB is the optimal blocksize for CSYTRF_AA.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, so the solution could not be computed.

Related Information

For this routine in other precisions, please see *dsysv_aa*, *ssysv_aa* and *zsysv_aa*. It also exists with a native C interface as *LAPACKE_csysv_aa*.

4.6.44 csysv_rk

CSYSV_RK computes the solution to a complex system of linear equations A * X = B, where A is an N-by-N symmetric matrix and X and B are N-by-NRHS matrices.

The bounded Bunch-Kaufman (rook) diagonal pivoting method is used to factor A as

```
A = P*U*D*(U**T)*(P**T),  if UPLO = 'U', or
A = P*L*D*(L**T)*(P**T),  if UPLO = 'L',
```

where U (or L) is unit upper (or lower) triangular matrix, U^T (or L^T) is the transpose of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

CSYTRF_RK is called to compute the factorization of a complex symmetric matrix. The factored form of A is then used to solve the system of equations A * X = B by calling BLAS3 routine CSYTRS_3.

Syntax

Fortran specification:

```
use armpl_library

subroutine csysv_rk(UPLO, N, NRHS, A, LDA, E, IPIV, B, LDB, WORK, LWORK,
                    INFO)
```

C specification:

```
#include "armpl.h"

void csysv_rk_(const char *uplo, const armpl_int_t *n,
               const armpl_int_t *nrhs, armpl_singlecomplex_t *a,
               const armpl_int_t *lda, armpl_singlecomplex_t *e,
               armpl_int_t *ipiv, armpl_singlecomplex_t *b,
               const armpl_int_t *ldb, armpl_singlecomplex_t *work,
               const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U': the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced.

If UPLO = 'L': the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, diagonal of the block diagonal matrix D and factors U or L as computed by CSYTRF_RK: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

For more info see the description of CSYTRF_RK routine.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Output parameter.

E is COMPLEX

E is an array, dimension (N). On exit, contains the output computed by the factorization routine CSYTRF_RK, i.e. the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) is set to 0; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) is set to 0.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is set to 0 in both UPLO = 'U' or UPLO = 'L' cases.

For more info see the description of CSYTRF_RK routine.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D, as determined by CSYTRF_RK.

For more info see the description of CSYTRF_RK routine.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)).. Work array used in the factorization stage. On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= 1. For best performance of factorization stage LWORK >= max(1,N*NB), where NB is the optimal blocksize for CSYTRF_RK.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array for factorization stage, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit

< 0: If INFO = -k, the k-th argument had an illegal value

> 0: If INFO = k, the matrix A is singular, because: If UPLO = 'U': column k in the upper triangular part of A contains all zeros. If UPLO = 'L': column k in the lower triangular part of A contains all zeros.

Therefore D(k,k) is exactly zero, and superdiagonal elements of column k of U (or subdiagonal elements of column k of L) are all zeros. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

NOTE: INFO only stores the first occurrence of a singularity, any subsequent occurrence of singularity is not stored in INFO even though the factorization always completes.

Related Information

For this routine in other precisions, please see *dsysv_rk*, *ssysv_rk* and *zsysv_rk*. It also exists with a native C interface as *LAPACKE_csysv_rk*.

4.6.45 csysv_rook

CSYSV_ROOK computes the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N symmetric matrix and X and B are N-by-NRHS matrices.

The diagonal pivoting method is used to factor A as

```
A = U * D * U**T,  if UPLO = 'U', or
A = L * D * L**T,  if UPLO = 'L',
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

CSYTRF_ROOK is called to compute the factorization of a complex symmetric matrix A using the bounded Bunch-Kaufman ("rook") diagonal pivoting method.

The factored form of A is then used to solve the system of equations A * X = B by calling CSYTRS_ROOK.

Syntax

Fortran specification:

```
use armpl_library

subroutine csysv_rook(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void csysv_rook_(const char *uplo, const armpl_int_t *n,
                 const armpl_int_t *nrhs, armpl_singlecomplex_t *a,
                 const armpl_int_t *lda, armpl_int_t *ipiv,
                 armpl_singlecomplex_t *b, const armpl_int_t *ldb,
                 armpl_singlecomplex_t *work, const armpl_int_t *lwork,
                 armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by CSYTRF_ROOK.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D, as determined by CSYTRF_ROOK.

If UPLO = 'U': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k-1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k-1 and -IPIV(k-1) were inerchaged, D(k-1:k,k-1:k) is a 2-by-2 diagonal block.

If UPLO = 'L': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k+1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k+1 and -IPIV(k+1) were inerchaged, D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= 1, and for best performance LWORK >= max(1,N*NB), where NB is the optimal blocksize for CSYTRF_ROOK.

TRS will be done with Level 2 BLAS

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, so the solution could not be computed.

Related Information

For this routine in other precisions, please see *dsysv_rook*, *ssysv_rook* and *zsysv_rook*. It also exists with a native C interface as *LAPACKE_csysv_rook*.

4.6.46 csysvx

csysvx uses the diagonal pivoting factorization to compute the solution to a complex system of linear equations A * X = B, where A is an N-by-N symmetric matrix and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed:
```

1. If FACT = 'N', the diagonal pivoting method is used to factor A.

```
 The form of the factorization is
   A = U * D * U**T,  if UPLO = 'U', or
   A = L * D * L**T,  if UPLO = 'L',
where U (or L) is a product of permutation and unit upper (lower)
triangular matrices, and D is symmetric and block diagonal with
1-by-1 and 2-by-2 diagonal blocks.
```

2. If some D(i,i)=0, so that D is exactly singular, then the routine

```
returns with INFO = i. Otherwise, the factored form of A is used
to estimate the condition number of the matrix A.  If the
reciprocal of the condition number is less than machine precision,
INFO = N+1 is returned as a warning, but the routine still goes on
to solve for X and compute error bounds as described below.
```

3. The system of equations is solved for X using the factored form

```
of A.
```

4. Iterative refinement is applied to improve the computed solution

```
matrix and calculate error bounds and backward error estimates
for it.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine csysvx(FACT, UPLO, N, NRHS, A, LDA, AF, LDAF, IPIV, B, LDB, X, LDX,
                  RCOND, FERR, BERR, WORK, LWORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void csysvx_(const char *fact, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *nrhs, const armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_singlecomplex_t *af,
             const armpl_int_t *ldaf, armpl_int_t *ipiv,
             const armpl_singlecomplex_t *b, const armpl_int_t *ldb,
             armpl_singlecomplex_t *x, const armpl_int_t *ldx, float *rcond,
             float *ferr, float *berr, armpl_singlecomplex_t *work,
             const armpl_int_t *lwork, float *rwork, armpl_int_t *info,
             ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of A has been supplied on entry. = 'F': On entry, AF and IPIV contain the factored form of A. A, AF and IPIV will not be modified. = 'N': The matrix A will be copied to AF and factored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input and output parameter.

AF is COMPLEX

AF is an array, dimension (LDAF, N). If FACT = 'F', then AF is an input argument and on entry contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by CSYTRF.

If FACT = 'N', then AF is an output argument and on exit returns the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T .

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains details of the interchanges and the block structure of D, as determined by CSYTRF. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

If FACT = 'N', then IPIV is an output argument and on exit contains details of the interchanges and the block structure of D, as determined by CSYTRF.

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). The N-by-NRHS right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is COMPLEX

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

The estimate of the reciprocal condition number of the matrix A. If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= max(1,2*N), and for best performance, when FACT = 'N', LWORK >= max(1,2*N,N*NB), where NB is the optimal blocksize for CSYTRF.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: D(i,i) is exactly zero. The factorization has been completed but the factor D is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+1: D is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *dsysvx*, *ssysvx* and *zsysvx*. It also exists with a native C interface as *LAPACKE_csysvx*.

4.6.47 csysvxx

```
CSYSVXX uses the diagonal pivoting factorization to compute the
   solution to a complex system of linear equations A * X = B, where
   A is an N-by-N symmetric matrix and X and B are N-by-NRHS
   matrices.

   If requested, both normwise and maximum componentwise error bounds
   are returned. CSYSVXX will return a solution with a tiny
   guaranteed error (O(eps) where eps is the working machine
   precision) unless the matrix is very ill-conditioned, in which
   case a warning is returned. Relevant condition numbers also are
   calculated and returned.

   CSYSVXX accepts user-provided factorizations and equilibration
   factors; see the definitions of the FACT and EQUED options.
   Solving with refinement and using a factorization from a previous
   CSYSVXX call will also produce a solution with either O(eps)
   errors or warnings, but we cannot make that claim for general
   user-provided factorizations and equilibration factors if they
   differ from what CSYSVXX would itself produce.

                                                                  The␣
↪following steps are performed:

   1. If FACT = 'E', real scaling factors are computed to equilibrate
   the system:

     diag(S)*A*diag(S)     *inv(diag(S))*X = diag(S)*B

   Whether or not the system will be equilibrated depends on the
   scaling of the matrix A, but if equilibration is used, A is
   overwritten by diag(S)*A*diag(S) and B by diag(S)*B.

   2. If FACT = 'N' or 'E', the LU decomposition is used to factor
   the matrix A (after equilibration if FACT = 'E') as

      A = U * D * U**T,  if UPLO = 'U', or
      A = L * D * L**T,  if UPLO = 'L',

   where U (or L) is a product of permutation and unit upper (lower)
   triangular matrices, and D is symmetric and block diagonal with
   1-by-1 and 2-by-2 diagonal blocks.

   3. If some D(i,i)=0, so that D is exactly singular, then the
   routine returns with INFO = i. Otherwise, the factored form of A
   is used to estimate the condition number of the matrix A (see
   argument RCOND).  If the reciprocal of the condition number is
   less than machine precision, the routine still goes on to solve
   for X and compute error bounds as described below.

   4. The system of equations is solved for X using the factored form
   of A.

   5. By default (unless PARAMS(LA_LINRX_ITREF_I) is set to zero),
   the routine will use iterative refinement to try to get a small
```

(continues on next page)

(continued from previous page)

```
   error and error bounds.  Refinement calculates the residual to at
   least twice the working precision.

   6. If equilibration was used, the matrix X is premultiplied by
   diag(R) so that it solves the original system before
   equilibration.                          Some optional parameters are bundled␣
↪in the PARAMS array.  These
    settings determine how refinement is performed, but often the
    defaults are acceptable.  If the defaults are acceptable, users
    can pass NPARAMS = 0 which prevents the source code from accessing
    the PARAMS argument.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine csysvxx(FACT, UPLO, N, NRHS, A, LDA, AF, LDAF, IPIV, EQUED, S, B,
                   LDB, X, LDX, RCOND, RPVGRW, BERR, N_ERR_BNDS,
                   ERR_BNDS_NORM, ERR_BNDS_COMP, NPARAMS, PARAMS, WORK, RWORK,
                   INFO)
```

C specification:

```
#include "armpl.h"

void csysvxx_(const char *fact, char *uplo, const armpl_int_t *n,
              const armpl_int_t *nrhs, armpl_singlecomplex_t *a,
              const armpl_int_t *lda, armpl_singlecomplex_t *af,
              const armpl_int_t *ldaf, armpl_int_t *ipiv, char *equed,
              float *s, armpl_singlecomplex_t *b, const armpl_int_t *ldb,
              armpl_singlecomplex_t *x, const armpl_int_t *ldx, float *rcond,
              float *rpvgrw, float *berr, const armpl_int_t *n_err_bnds,
              float *err_bnds_norm, float *err_bnds_comp,
              const armpl_int_t *nparams, float *params,
              armpl_singlecomplex_t *work, float *rwork, armpl_int_t *info,
              ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AF and IPIV contain the factored form of A. If EQUED is not 'N', the matrix A has been equilibrated with scaling factors given by S. A, AF, and IPIV are not modified. = 'N': The matrix A will be copied to AF and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AF and factored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if FACT = 'E' and EQUED = 'Y', A is overwritten by diag(S)*A*diag(S).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input and output parameter.

AF is COMPLEX

AF is an array, dimension (LDAF, N). If FACT = 'F', then AF is an input argument and on entry contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by SSYTRF.

If FACT = 'N', then AF is an output argument and on exit returns the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T .

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains details of the interchanges and the block structure of D, as determined by SSYTRF. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

If FACT = 'N', then IPIV is an output argument and on exit contains details of the interchanges and the block structure of D, as determined by SSYTRF.

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'Y': Both row and column equilibration, i.e., A has been replaced by diag(S) * A * diag(S). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

S Input and output parameter.

S is REAL

S is an array, dimension (N). The scale factors for A. If EQUED = 'Y', A is multiplied on the left and right by diag(S). S is an input argument if FACT = 'F'; otherwise, S is an output argument. If FACT = 'F' and EQUED = 'Y', each element of S must be positive. If S is output, each element of S is a power of the radix. If S is input, each element of S should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or

overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if EQUED = 'N', B is not modified; if EQUED = 'Y', B is overwritten by diag(S)*B;

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is COMPLEX

X is an array, dimension (LDX, NRHS). If INFO = 0, the N-by-NRHS solution matrix X to the original system of equations. Note that A and B are modified on exit if EQUED .ne. 'N', and the solution to the equilibrated system is inv(diag(S))*X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

RPVGRW Output parameter.

RPVGRW is REAL

Reciprocal pivot growth. On exit, this contains the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If this is much less than 1, then the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, estimated condition numbers, and error bounds could be unreliable. If factorization fails with 0<INFO<=N, then this contains the reciprocal pivot growth factor for the leading INFO columns of A.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). Componentwise relative backward error. This is the componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

N_ERR_BNDS Input parameter.

N_ERR_BNDS is INTEGER

Number of error bounds to return for each right hand side and each type (normwise or componentwise). See ERR_BNDS_NORM and ERR_BNDS_COMP below.

ERR_BNDS_NORM Output parameter.

ERR_BNDS_NORM is REAL

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Output parameter.

ERR_BNDS_COMP is REAL

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:, N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

NPARAMS Input parameter.

NPARAMS is INTEGER

Specifies the number of parameters set in PARAMS. If .LE. 0, the PARAMS array is never referenced and default values are used.

PARAMS Input and output parameter.

PARAMS is REAL

PARAMS is an array, dimension NPARAMS. Specifies algorithm parameters. If an entry is .LT. 0.0, then that entry will be filled with default value used for that parameter. Only positions up to NPARAMS are accessed; defaults are used for higher-numbered parameters.

PARAMS(LA_LINRX_ITREF_I = 1) : Whether to perform iterative refinement or not. Default: 1.0 = 0.0 : No refinement is performed, and no error bounds are computed. = 1.0 : Use the double-precision refinement algorithm, possibly with doubled-single computations if the compilation environment does not support DOUBLE PRECISION. (other values are reserved for future use)

PARAMS(LA_LINRX_ITHRESH_I = 2) : Maximum number of residual computations allowed for refinement. Default: 10 Aggressive: Set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in err_bnds_norm and err_bnds_comp may no longer be trustworthy.

PARAMS(LA_LINRX_CWISE_I = 3) : Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Default: 1.0 (attempt componentwise convergence)

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. The solution to every right-hand side is guaranteed. < 0: If INFO = -i, the i-th argument had an illegal value > 0 and <= N: U(INFO, INFO) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+J: The solution corresponding to the Jth right-hand side is not guaranteed. The solutions corresponding to other right- hand sides K with K > J may not be guaranteed as well, but only the first such right-hand side is reported. If a small componentwise error is not requested (PARAMS(3) = 0.0) then the Jth right-hand side is the first with a normwise error bound that is not guaranteed (the smallest J such that ERR_BNDS_NORM(J,1) = 0.0). By default (PARAMS(3) = 1.0) the Jth right-hand side is the first with either a normwise or componentwise error bound that is not guaranteed (the smallest J such that either ERR_BNDS_NORM(J,1) = 0.0 or ERR_BNDS_COMP(J,1) = 0.0). See the definition of ERR_BNDS_NORM(:,1) and ERR_BNDS_COMP(:,1). To get information about all of the right-hand sides check ERR_BNDS_NORM or ERR_BNDS_COMP.

Related Information

For this routine in other precisions, please see *dsysvxx*, *ssysvxx* and *zsysvxx*. It also exists with a native C interface as *LAPACKE_csysvxx*.

4.6.48 csytrs

csytrs solves a system of linear equations A*X = B with a complex symmetric matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by CSYTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine csytrs(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void csytrs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const armpl_int_t *ipiv, armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CSYTRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CSYTRF.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dsytrs*, *ssytrs* and *zsytrs*. It also exists with a native C interface as *LAPACKE_csytrs*.

4.6.49 csytrs2

`csytrs2` solves a system of linear equations A*X = B with a COMPLEX symmetric matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by CSYTRF and converted by CSYCONV.

Syntax

Fortran specification:

```
use armpl_library

subroutine csytrs2(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void csytrs2_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
              const armpl_singlecomplex_t *a, const armpl_int_t *lda,
              const armpl_int_t *ipiv, armpl_singlecomplex_t *b,
              const armpl_int_t *ldb, armpl_singlecomplex_t *work,
              armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is `A` = U*D*U^T ; = 'L': Lower triangular, form is `A` = L*D*L^T .

N Input parameter.

`N` is `INTEGER`

The order of the matrix `A`. `N` >= 0.

NRHS Input parameter.

`NRHS` is `INTEGER`

The number of right hand sides, i.e., the number of columns of the matrix `B`. `NRHS` >= 0.

A Input and output parameter.

`A` is `COMPLEX`

`A` is an array, dimension (`LDA`, `N`). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CSYTRF. Note that `A` is input / output. This might be counter-intuitive, and one may think that `A` is input only. `A` is input / output. This is because, at the start of the subroutine, we permute `A` in a "better" form and then we permute `A` back to its original form at the end.

LDA Input parameter.

`LDA` is `INTEGER`

The leading dimension of the array `A`. `LDA` >= max(1, `N`).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CSYTRF.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dsytrs2*, *ssytrs2* and *zsytrs2*. It also exists with a native C interface as *LAPACKE_csytrs2*.

4.6.50 csytrs_3

CSYTRS_3 solves a system of linear equations A * X = B with a complex symmetric matrix A using the factorization computed by CSYTRF_RK or CSYTRF_BK:

```
A = P*U*D*(U**T)*(P**T) or A = P*L*D*(L**T)*(P**T),
```

where U (or L) is unit upper (or lower) triangular matrix, U^T (or L^T) is the transpose of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This algorithm is using Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine csytrs_3(UPLO, N, NRHS, A, LDA, E, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void csytrs_3_(const char *uplo, const armpl_int_t *n,
               const armpl_int_t *nrhs, const armpl_singlecomplex_t *a,
               const armpl_int_t *lda, const armpl_singlecomplex_t *e,
               const armpl_int_t *ipiv, armpl_singlecomplex_t *b,
               const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix: = 'U': Upper triangular, form is A = P*U*D*(U^T)*(P^T); = 'L': Lower triangular, form is A = P*L*D*(L^T)*(P^T).

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). Diagonal of the block diagonal matrix D and factors U or L as computed by CSYTRF_RK and CSYTRF_BK: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D should be provided on entry in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Input parameter.

E is COMPLEX

E is an array, dimension (N). On entry, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i),i=2:N, E(1) not referenced; If UPLO = 'L': E(i) = D(i+1,i),i=1:N-1, E(N) not referenced.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is not referenced in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CSYTRF_RK or CSYTRF_BK.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dsytrs_3*, *ssytrs_3* and *zsytrs_3*. It also exists with a native C interface as *LAPACKE_csytrs_3*.

4.6.51 csytrs_aa

CSYTRS_AA solves a system of linear equations A*X = B with a complex symmetric matrix A using the factorization A = U*T*U^T or A = L*T*L^T computed by CSYTRF_AA.

Syntax

Fortran specification:

```
use armpl_library

subroutine csytrs_aa(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void csytrs_aa_(const char *uplo, const armpl_int_t *n,
                const armpl_int_t *nrhs, const armpl_singlecomplex_t *a,
                const armpl_int_t *lda, const armpl_int_t *ipiv,
                armpl_singlecomplex_t *b, const armpl_int_t *ldb,
                const armpl_singlecomplex_t *work, const armpl_int_t *lwork,
                armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*T*U^T ; = 'L': Lower triangular, form is A = L*T*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). Details of factors computed by CSYTRF_AA.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges as computed by CSYTRF_AA.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Input parameter.

WORK is DOUBLE array, dimension (MAX(1, LWORK))

LWORK Input parameter.

LWORK is INTEGER, LWORK >= MAX(1,3*N-2).

param[out] INFO verbatim INFO is INTEGER = 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dsytrs_aa*, *ssytrs_aa* and *zsytrs_aa*. It also exists with a native C interface as *LAPACKE_csytrs_aa*.

4.6.52 csytrs_rook

CSYTRS_ROOK solves a system of linear equations A*X = B with a complex symmetric matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by CSYTRF_ROOK.

Syntax

Fortran specification:

```
use armpl_library

subroutine csytrs_rook(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void csytrs_rook_(const char *uplo, const armpl_int_t *n,
                  const armpl_int_t *nrhs, const armpl_singlecomplex_t *a,
                  const armpl_int_t *lda, const armpl_int_t *ipiv,
                  armpl_singlecomplex_t *b, const armpl_int_t *ldb,
                  armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CSYTRF_ROOK.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CSYTRF_ROOK.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dsytrs_rook*, *ssytrs_rook* and *zsytrs_rook*. It also exists with a native C interface as *LAPACKE_csytrs_rook*.

4.6.53 ctbtrs

ctbtrs solves a triangular system of the form

```
A * X = B,  A**T * X = B,  or  A**H * X = B,
```

where A is a triangular band matrix of order N, and B is an N-by-NRHS matrix. A check is made to verify that A is nonsingular.

Syntax

Fortran specification:

```
use armpl_library

subroutine ctbtrs(UPLO, TRANS, DIAG, N, KD, NRHS, AB, LDAB, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void ctbtrs_(const char *uplo, const char *trans, const char *diag,
             const armpl_int_t *n, const armpl_int_t *kd,
             const armpl_int_t *nrhs, const armpl_singlecomplex_t *ab,
             const armpl_int_t *ldab, armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose)

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals or subdiagonals of the triangular band matrix A. KD >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AB Input parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). The upper or lower triangular band matrix A, stored in the first kd+1 rows of AB. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd). If DIAG = 'U', the diagonal elements of A are not referenced and are assumed to be 1.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, if INFO = 0, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element of A is zero, indicating that the matrix is singular and the solutions X have not been computed.

Related Information

For this routine in other precisions, please see *dtbtrs*, *stbtrs* and *ztbtrs*. It also exists with a native C interface as *LAPACKE_ctbtrs*.

4.6.54 ctptrs

ctptrs solves a triangular system of the form

```
A * X = B,  A**T * X = B,  or  A**H * X = B,
```

where A is a triangular matrix of order N stored in packed format, and B is an N-by-NRHS matrix. A check is made to verify that A is nonsingular.

Syntax

Fortran specification:

```
use armpl_library

subroutine ctptrs(UPLO, TRANS, DIAG, N, NRHS, AP, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void ctptrs_(const char *uplo, const char *trans, const char *diag,
             const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_singlecomplex_t *ap, armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose)

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AP Input parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). The upper or lower triangular matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, if INFO = 0, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element of A is zero, indicating that the matrix is singular and the solutions X have not been computed.

Related Information

For this routine in other precisions, please see *dtptrs*, *stptrs* and *ztptrs*. It also exists with a native C interface as *LAPACKE_ctptrs*.

4.6.55 dgbsv

dgbsv computes the solution to a real system of linear equations A * X = B, where A is a band matrix of order N with KL subdiagonals and KU superdiagonals, and X and B are N-by-NRHS matrices.

The LU decomposition with partial pivoting and row interchanges is used to factor A as A = L * U, where L is a product of permutation and unit lower triangular matrices with KL subdiagonals, and U is upper triangular with KL+KU superdiagonals. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgbsv(N, KL, KU, NRHS, AB, LDAB, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void dgbsv_(const armpl_int_t *n, const armpl_int_t *kl,
            const armpl_int_t *ku, const armpl_int_t *nrhs, double *ab,
            const armpl_int_t *ldab, armpl_int_t *ipiv, double *b,
            const armpl_int_t *ldb, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AB Input and output parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows KL+1 to 2*KL+KU+1; rows 1 to KL of the array need not be set. The j-th column of A is stored in the j-th column of the array AB as follows: AB(KL+KU+1+i-j,j) = A(i,j) for max(1,j-KU)<=i<=min(N,j+KL) On exit, details of the factorization: U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1. See below for further details.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= 2*KL+KU+1.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices that define the permutation matrix P; row i of the matrix was interchanged with row IPIV(i).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and the solution has not been computed.

Related Information

For this routine in other precisions, please see *cgbsv*, *sgbsv* and *zgbsv*. It also exists with a native C interface as *LAPACKE_dgbsv*.

4.6.56 dgbsvx

dgbsvx uses the LU factorization to compute the solution to a real system of linear equations A * X = B, A^T * X = B, or A^H * X = B, where A is a band matrix of order N with KL subdiagonals and KU superdiagonals, and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed by this subroutine:
```

1. If FACT = 'E', real scaling factors are computed to equilibrate

```
 the system:
   TRANS = 'N':  diag(R)*A*diag(C)      *inv(diag(C))*X = diag(R)*B
   TRANS = 'T': (diag(R)*A*diag(C))**T  *inv(diag(R))*X = diag(C)*B
   TRANS = 'C': (diag(R)*A*diag(C))**H  *inv(diag(R))*X = diag(C)*B
Whether or not the system will be equilibrated depends on the
scaling of the matrix A, but if equilibration is used, A is
overwritten by diag(R)*A*diag(C) and B by diag(R)*B (if TRANS='N')
or diag(C)*B (if TRANS = 'T' or 'C').
```

2. If FACT = 'N' or 'E', the LU decomposition is used to factor the

```
matrix A (after equilibration if FACT = 'E') as
   A = L * U,
where L is a product of permutation and unit lower triangular
```

(continues on next page)

(continued from previous page)

```
matrices with KL subdiagonals, and U is upper triangular with
KL+KU superdiagonals.
```

3. If some U(i,i)=0, so that U is exactly singular, then the routine

```
returns with INFO = i. Otherwise, the factored form of A is used
to estimate the condition number of the matrix A.  If the
reciprocal of the condition number is less than machine precision,
INFO = N+1 is returned as a warning, but the routine still goes on
to solve for X and compute error bounds as described below.
```

4. The system of equations is solved for X using the factored form

```
of A.
```

5. Iterative refinement is applied to improve the computed solution

```
matrix and calculate error bounds and backward error estimates
for it.
```

6. If equilibration was used, the matrix X is premultiplied by

```
 diag(C) (if TRANS = 'N') or diag(R) (if TRANS = 'T' or 'C') so
that it solves the original system before equilibration.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dgbsvx(FACT, TRANS, N, KL, KU, NRHS, AB, LDAB, AFB, LDAFB, IPIV,
                  EQUED, R, C, B, LDB, X, LDX, RCOND, FERR, BERR, WORK, IWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dgbsvx_(const char *fact, const char *trans, const armpl_int_t *n,
             const armpl_int_t *kl, const armpl_int_t *ku,
             const armpl_int_t *nrhs, double *ab, const armpl_int_t *ldab,
             double *afb, const armpl_int_t *ldafb, armpl_int_t *ipiv,
             char *equed, double *r, double *c, double *b,
             const armpl_int_t *ldb, double *x, const armpl_int_t *ldx,
             double *rcond, double *ferr, double *berr, double *work,
             armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AFB and IPIV contain the factored form of A. If EQUED is not 'N', the matrix A has been equilibrated with scaling factors given by R and C. AB, AFB, and IPIV are not modified. = 'N': The matrix A will be copied to AFB and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AFB and factored.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations. = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Transpose)

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AB Input and output parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(KU+1+i-j,j) = A(i,j) for max(1,j-KU)<=i<=min(N,j+kl)

If FACT = 'F' and EQUED is not 'N', then A must have been equilibrated by the scaling factors in R and/or C. AB is not modified if FACT = 'F' or 'N', or if FACT = 'E' and EQUED = 'N' on exit.

On exit, if EQUED .ne. 'N', A is scaled as follows: EQUED = 'R': A := diag(R) * A EQUED = 'C': A := A * diag(C) EQUED = 'B': A := diag(R) * A * diag(C).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KL+KU+1.

AFB Input and output parameter.

AFB is DOUBLE PRECISION

AFB is an array, dimension (LDAFB, N). If FACT = 'F', then AFB is an input argument and on entry contains details of the LU factorization of the band matrix A, as computed by DGBTRF. U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1. If EQUED .ne. 'N', then AFB is the factored form of the equilibrated matrix A.

If FACT = 'N', then AFB is an output argument and on exit returns details of the LU factorization of A.

If FACT = 'E', then AFB is an output argument and on exit returns details of the LU factorization of the equilibrated matrix A (see the description of AB for the form of the equilibrated matrix).

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AFB. LDAFB >= 2*KL+KU+1.

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains the pivot indices from the factorization A = L*U as computed by DGBTRF; row i of the matrix was interchanged with row IPIV(i).

If FACT = 'N', then IPIV is an output argument and on exit contains the pivot indices from the factorization A = L*U of the original matrix A.

If FACT = 'E', then IPIV is an output argument and on exit contains the pivot indices from the factorization A = L*U of the equilibrated matrix A.

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'R': Row equilibration, i.e., A has been premultiplied by diag(R). = 'C': Column equilibration, i.e., A has been postmultiplied by diag(C). = 'B': Both row and column equilibration, i.e., A has been replaced by diag(R) * A * diag(C). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

R Input and output parameter.

R is DOUBLE PRECISION

R is an array, dimension (N). The row scale factors for A. If EQUED = 'R' or 'B', A is multiplied on the left by diag(R); if EQUED = 'N' or 'C', R is not accessed. R is an input argument if FACT = 'F'; otherwise, R is an output argument. If FACT = 'F' and EQUED = 'R' or 'B', each element of R must be positive.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). The column scale factors for A. If EQUED = 'C' or 'B', A is multiplied on the right by diag(C); if EQUED = 'N' or 'R', C is not accessed. C is an input argument if FACT = 'F'; otherwise, C is an output argument. If FACT = 'F' and EQUED = 'C' or 'B', each element of C must be positive.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, if EQUED = 'N', B is not modified; if TRANS = 'N' and EQUED = 'R' or 'B', B is overwritten by diag(R)*B; if TRANS = 'T' or 'C' and EQUED = 'C' or 'B', B is overwritten by diag(C)*B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X to the original system of equations. Note that A and B are modified on exit if EQUED .ne. 'N', and the solution to the equilibrated system is inv(diag(C))* X if TRANS = 'N' and EQUED = 'C' or 'B', or inv(diag(R))* X if TRANS = 'T' or 'C' and EQUED = 'R' or 'B'.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The estimate of the reciprocal condition number of the matrix A after equilibration (if done). If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (3*N). On exit, WORK(1) contains the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If WORK(1) is much less than 1, then the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, condition estimator RCOND, and forward error bound FERR could be unreliable. If factorization fails with 0<INFO<=N, then WORK(1) contains the reciprocal pivot growth factor for the leading INFO columns of A.

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+1: U is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *cgbsvx*, *sgbsvx* and *zgbsvx*. It also exists with a native C interface as *LAPACKE_dgbsvx*.

4.6.57 dgbsvxx

```
DGBSVXX uses the LU factorization to compute the solution to a
   double precision system of linear equations  A * X = B,  where A is an
   N-by-N matrix and X and B are N-by-NRHS matrices.

   If requested, both normwise and maximum componentwise error bounds
   are returned. DGBSVXX will return a solution with a tiny
   guaranteed error (O(eps) where eps is the working machine
   precision) unless the matrix is very ill-conditioned, in which
   case a warning is returned. Relevant condition numbers also are
```

(continues on next page)

(continued from previous page)

```
   calculated and returned.

   DGBSVXX accepts user-provided factorizations and equilibration
   factors; see the definitions of the FACT and EQUED options.
   Solving with refinement and using a factorization from a previous
   DGBSVXX call will also produce a solution with either O(eps)
   errors or warnings, but we cannot make that claim for general
   user-provided factorizations and equilibration factors if they
   differ from what DGBSVXX would itself produce.

                                                                  The␣
↪following steps are performed:

   1. If FACT = 'E', double precision scaling factors are computed to equilibrate
   the system:

     TRANS = 'N':  diag(R)*A*diag(C)     *inv(diag(C))*X = diag(R)*B
     TRANS = 'T': (diag(R)*A*diag(C))**T *inv(diag(R))*X = diag(C)*B
     TRANS = 'C': (diag(R)*A*diag(C))**H *inv(diag(R))*X = diag(C)*B

   Whether or not the system will be equilibrated depends on the
   scaling of the matrix A, but if equilibration is used, A is
   overwritten by diag(R)*A*diag(C) and B by diag(R)*B (if TRANS='N')
   or diag(C)*B (if TRANS = 'T' or 'C').

   2. If FACT = 'N' or 'E', the LU decomposition is used to factor
   the matrix A (after equilibration if FACT = 'E') as

     A = P * L * U,

   where P is a permutation matrix, L is a unit lower triangular
   matrix, and U is upper triangular.

   3. If some U(i,i)=0, so that U is exactly singular, then the
   routine returns with INFO = i. Otherwise, the factored form of A
   is used to estimate the condition number of the matrix A (see
   argument RCOND). If the reciprocal of the condition number is less
   than machine precision, the routine still goes on to solve for X
   and compute error bounds as described below.

   4. The system of equations is solved for X using the factored form
   of A.

   5. By default (unless PARAMS(LA_LINRX_ITREF_I) is set to zero),
   the routine will use iterative refinement to try to get a small
   error and error bounds.  Refinement calculates the residual to at
   least twice the working precision.

   6. If equilibration was used, the matrix X is premultiplied by
   diag(C) (if TRANS = 'N') or diag(R) (if TRANS = 'T' or 'C') so
   that it solves the original system before equilibration.
↪   Some optional parameters are bundled in the PARAMS array.  These
    settings determine how refinement is performed, but often the
    defaults are acceptable.  If the defaults are acceptable, users
    can pass NPARAMS = 0 which prevents the source code from accessing
    the PARAMS argument.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dgbsvxx(FACT, TRANS, N, KL, KU, NRHS, AB, LDAB, AFB, LDAFB, IPIV,
                   EQUED, R, C, B, LDB, X, LDX, RCOND, RPVGRW, BERR,
                   N_ERR_BNDS, ERR_BNDS_NORM, ERR_BNDS_COMP, NPARAMS, PARAMS,
                   WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgbsvxx_(const char *fact, const char *trans, const armpl_int_t *n,
              const armpl_int_t *kl, const armpl_int_t *ku,
              const armpl_int_t *nrhs, double *ab, const armpl_int_t *ldab,
              double *afb, const armpl_int_t *ldafb, armpl_int_t *ipiv,
              char *equed, double *r, double *c, double *b,
              const armpl_int_t *ldb, double *x, const armpl_int_t *ldx,
              double *rcond, double *rpvgrw, double *berr,
              const armpl_int_t *n_err_bnds, double *err_bnds_norm,
              double *err_bnds_comp, const armpl_int_t *nparams,
              double *params, double *work, armpl_int_t *iwork,
              armpl_int_t *info, ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AF and IPIV contain the factored form of A. If EQUED is not 'N', the matrix A has been equilibrated with scaling factors given by R and C. A, AF, and IPIV are not modified. = 'N': The matrix A will be copied to AF and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AF and factored.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate Transpose = Transpose)

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AB Input and output parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(KU+1+i-j,j) = A(i,j) for max(1,j-KU)<=i<=min(N,j+kl)

If FACT = 'F' and EQUED is not 'N', then AB must have been equilibrated by the scaling factors in R and/or C. AB is not modified if FACT = 'F' or 'N', or if FACT = 'E' and EQUED = 'N' on exit.

On exit, if EQUED .ne. 'N', A is scaled as follows: EQUED = 'R': A := diag(R) * A EQUED = 'C': A := A * diag(C) EQUED = 'B': A := diag(R) * A * diag(C).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KL+KU+1.

AFB Input and output parameter.

AFB is DOUBLE PRECISION

AFB is an array, dimension (LDAFB, N). If FACT = 'F', then AFB is an input argument and on entry contains details of the LU factorization of the band matrix A, as computed by DGBTRF. U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1. If EQUED .ne. 'N', then AFB is the factored form of the equilibrated matrix A.

If FACT = 'N', then AF is an output argument and on exit returns the factors L and U from the factorization A = P*L*U of the original matrix A.

If FACT = 'E', then AF is an output argument and on exit returns the factors L and U from the factorization A = P*L*U of the equilibrated matrix A (see the description of A for the form of the equilibrated matrix).

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AFB. LDAFB >= 2*KL+KU+1.

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains the pivot indices from the factorization A = P*L*U as computed by DGETRF; row i of the matrix was interchanged with row IPIV(i).

If FACT = 'N', then IPIV is an output argument and on exit contains the pivot indices from the factorization A = P*L*U of the original matrix A.

If FACT = 'E', then IPIV is an output argument and on exit contains the pivot indices from the factorization A = P*L*U of the equilibrated matrix A.

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'R': Row equilibration, i.e., A has been premultiplied by diag(R). = 'C': Column equilibration, i.e., A has been postmultiplied by diag(C). = 'B': Both row and column equilibration, i.e., A has been replaced by diag(R) * A * diag(C). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

R Input and output parameter.

R is DOUBLE PRECISION

R is an array, dimension (N). The row scale factors for A. If EQUED = 'R' or 'B', A is multiplied on the left by diag(R); if EQUED = 'N' or 'C', R is not accessed. R is an input argument if FACT = 'F'; otherwise, R is an output argument. If FACT = 'F' and EQUED = 'R' or 'B', each element of R must be positive. If R

is output, each element of R is a power of the radix. If R is input, each element of R should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). The column scale factors for A. If EQUED = 'C' or 'B', A is multiplied on the right by diag(C); if EQUED = 'N' or 'R', C is not accessed. C is an input argument if FACT = 'F'; otherwise, C is an output argument. If FACT = 'F' and EQUED = 'C' or 'B', each element of C must be positive. If C is output, each element of C is a power of the radix. If C is input, each element of C should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if EQUED = 'N', B is not modified; if TRANS = 'N' and EQUED = 'R' or 'B', B is overwritten by diag(R)*B; if TRANS = 'T' or 'C' and EQUED = 'C' or 'B', B is overwritten by diag(C)*B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, NRHS). If INFO = 0, the N-by-NRHS solution matrix X to the original system of equations. Note that A and B are modified on exit if EQUED .ne. 'N', and the solution to the equilibrated system is inv(diag(C))* X if TRANS = 'N' and EQUED = 'C' or 'B', or inv(diag(R))* X if TRANS = 'T' or 'C' and EQUED = 'R' or 'B'.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

RPVGRW Output parameter.

RPVGRW is DOUBLE PRECISION

Reciprocal pivot growth. On exit, this contains the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If this is much less than 1, then the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, estimated condition numbers, and error bounds could be unreliable. If factorization fails with 0<INFO<=N, then this contains the reciprocal pivot growth factor for the leading INFO columns of A. In DGESVX, this quantity is returned in WORK(1).

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). Componentwise relative backward error. This is the componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

N_ERR_BNDS Input parameter.

N_ERR_BNDS is INTEGER

Number of error bounds to return for each right hand side and each type (normwise or componentwise). See ERR_BNDS_NORM and ERR_BNDS_COMP below.

ERR_BNDS_NORM Output parameter.

ERR_BNDS_NORM is DOUBLE PRECISION

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * dlamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * dlamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * dlamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Output parameter.

ERR_BNDS_COMP is DOUBLE PRECISION

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:, N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * dlamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * dlamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * dlamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

NPARAMS Input parameter.

NPARAMS is INTEGER

Specifies the number of parameters set in PARAMS. If .LE. 0, the PARAMS array is never referenced and default values are used.

PARAMS Input and output parameter.

PARAMS is DOUBLE PRECISION

PARAMS is an array, dimension (NPARAMS). Specifies algorithm parameters. If an entry is .LT. 0.0, then that entry will be filled with default value used for that parameter. Only positions up to NPARAMS are accessed; defaults are used for higher-numbered parameters.

PARAMS(LA_LINRX_ITREF_I = 1) : Whether to perform iterative refinement or not. Default: 1.0D+0 = 0.0 : No refinement is performed, and no error bounds are computed. = 1.0 : Use the extra-precise refinement algorithm. (other values are reserved for future use)

PARAMS(LA_LINRX_ITHRESH_I = 2) : Maximum number of residual computations allowed for refinement. Default: 10 Aggressive: Set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in err_bnds_norm and err_bnds_comp may no longer be trustworthy.

PARAMS(LA_LINRX_CWISE_I = 3) : Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Default: 1.0 (attempt componentwise convergence)

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (4*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. The solution to every right-hand side is guaranteed. < 0: If INFO = -i, the i-th argument had an illegal value > 0 and <= N: U(INFO, INFO) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+J: The solution corresponding to the Jth right-hand side is not guaranteed. The solutions corresponding to other right- hand sides K with K > J may not be guaranteed as well, but only the first such right-hand side is reported. If a small componentwise error is not requested (PARAMS(3) = 0.0) then the Jth right-hand side is the first with a normwise error bound that is not guaranteed (the smallest J such that ERR_BNDS_NORM(J,1) = 0.0). By default (PARAMS(3) = 1.0) the Jth right-hand side is the first with either a normwise or componentwise error bound that is not guaranteed (the smallest J such that either ERR_BNDS_NORM(J,1) = 0.0 or ERR_BNDS_COMP(J,1) = 0.0). See the definition of ERR_BNDS_NORM(:,1) and ERR_BNDS_COMP(:,1). To get information about all of the right-hand sides check ERR_BNDS_NORM or ERR_BNDS_COMP.

Related Information

For this routine in other precisions, please see *cgbsvxx*, *sgbsvxx* and *zgbsvxx*. It also exists with a native C interface as *LAPACKE_dgbsvxx*.

4.6.58 dgbtrs

`dgbtrs` solves a system of linear equations

```
A * X = B  or  A**T * X = B
```

with a general band matrix A using the LU factorization computed by DGBTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgbtrs(TRANS, N, KL, KU, NRHS, AB, LDAB, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void dgbtrs_(const char *trans, const armpl_int_t *n, const armpl_int_t *kl,
             const armpl_int_t *ku, const armpl_int_t *nrhs, const double *ab,
             const armpl_int_t *ldab, const armpl_int_t *ipiv, double *b,
             const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

`TRANS` is `CHARACTER*1`

Specifies the form of the system of equations. = 'N': A * X = `B` (No transpose) = 'T': A^T * X = `B` (Transpose) = 'C': A^T * X = `B` (Conjugate transpose = Transpose)

N Input parameter.

`N` is `INTEGER`

The order of the matrix A. `N` >= 0.

KL Input parameter.

`KL` is `INTEGER`

The number of subdiagonals within the band of A. `KL` >= 0.

KU Input parameter.

`KU` is `INTEGER`

The number of superdiagonals within the band of A. `KU` >= 0.

NRHS Input parameter.

`NRHS` is `INTEGER`

The number of right hand sides, i.e., the number of columns of the matrix `B`. `NRHS` >= 0.

AB Input parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). Details of the LU factorization of the band matrix A, as computed by DGBTRF. U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= 2*KL+KU+1.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices; for 1 <= i <= N, row i of the matrix was interchanged with row IPIV(i).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgbtrs*, *sgbtrs* and *zgbtrs*. It also exists with a native C interface as *LAPACKE_dgbtrs*.

4.6.59 dgesv

dgesv computes the solution to a real system of linear equations

```
A * X = B,
```

where A is an N-by-N matrix and X and B are N-by-NRHS matrices.

The LU decomposition with partial pivoting and row interchanges is used to factor A as

```
A = P * L * U,
```

where P is a permutation matrix, L is unit lower triangular, and U is upper triangular. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgesv(N, NRHS, A, LDA, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void dgesv_(const armpl_int_t *n, const armpl_int_t *nrhs, double *a,
            const armpl_int_t *lda, armpl_int_t *ipiv, double *b,
            const armpl_int_t *ldb, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the N-by-N coefficient matrix A. On exit, the factors L and U from the factorization A = P*L*U; the unit diagonal elements of L are not stored.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices that define the permutation matrix P; row i of the matrix was interchanged with row IPIV(i).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

Related Information

For this routine in other precisions, please see *cgesv*, *sgesv* and *zgesv*. It also exists with a native C interface as *LAPACKE_dgesv*.

4.6.60 dgesvx

dgesvx uses the LU factorization to compute the solution to a real system of linear equations

```
A * X = B,
```

where A is an N-by-N matrix and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed:
```

1. If FACT = 'E', real scaling factors are computed to equilibrate

```
 the system:
   TRANS = 'N':  diag(R)*A*diag(C)      *inv(diag(C))*X = diag(R)*B
   TRANS = 'T': (diag(R)*A*diag(C))**T *inv(diag(R))*X = diag(C)*B
   TRANS = 'C': (diag(R)*A*diag(C))**H *inv(diag(R))*X = diag(C)*B
Whether or not the system will be equilibrated depends on the
scaling of the matrix A, but if equilibration is used, A is
overwritten by diag(R)*A*diag(C) and B by diag(R)*B (if TRANS='N')
or diag(C)*B (if TRANS = 'T' or 'C').
```

2. If FACT = 'N' or 'E', the LU decomposition is used to factor the

```
 matrix A (after equilibration if FACT = 'E') as
   A = P * L * U,
where P is a permutation matrix, L is a unit lower triangular
matrix, and U is upper triangular.
```

3. If some U(i,i)=0, so that U is exactly singular, then the routine

```
 returns with INFO = i. Otherwise, the factored form of A is used
to estimate the condition number of the matrix A.  If the
reciprocal of the condition number is less than machine precision,
INFO = N+1 is returned as a warning, but the routine still goes on
to solve for X and compute error bounds as described below.
```

4. The system of equations is solved for X using the factored form

```
of A.
```

5. Iterative refinement is applied to improve the computed solution

```
 matrix and calculate error bounds and backward error estimates
for it.
```

6. If equilibration was used, the matrix X is premultiplied by

```
 diag(C) (if TRANS = 'N') or diag(R) (if TRANS = 'T' or 'C') so
that it solves the original system before equilibration.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dgesvx(FACT, TRANS, N, NRHS, A, LDA, AF, LDAF, IPIV, EQUED, R, C,
                  B, LDB, X, LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgesvx_(const char *fact, const char *trans, const armpl_int_t *n,
             const armpl_int_t *nrhs, double *a, const armpl_int_t *lda,
             double *af, const armpl_int_t *ldaf, armpl_int_t *ipiv,
             char *equed, double *r, double *c, double *b,
             const armpl_int_t *ldb, double *x, const armpl_int_t *ldx,
             double *rcond, double *ferr, double *berr, double *work,
             armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AF and IPIV contain the factored form of A. If EQUED is not 'N', the matrix A has been equilibrated with scaling factors given by R and C. A, AF, and IPIV are not modified. = 'N': The matrix A will be copied to AF and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AF and factored.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Transpose)

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A. If FACT = 'F' and EQUED is not 'N', then A must have been equilibrated by the scaling factors in R and/or C. A is not modified if FACT = 'F' or 'N', or if FACT = 'E' and EQUED = 'N' on exit.

On exit, if EQUED .ne. 'N', A is scaled as follows: EQUED = 'R': A := diag(R) * A EQUED = 'C': A := A * diag(C) EQUED = 'B': A := diag(R) * A * diag(C).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input and output parameter.

AF is DOUBLE PRECISION

AF is an array, dimension (LDAF, N). If FACT = 'F', then AF is an input argument and on entry contains the factors L and U from the factorization A = P*L*U as computed by DGETRF. If EQUED .ne. 'N', then AF is the factored form of the equilibrated matrix A.

If FACT = 'N', then AF is an output argument and on exit returns the factors L and U from the factorization A = P*L*U of the original matrix A.

If FACT = 'E', then AF is an output argument and on exit returns the factors L and U from the factorization A = P*L*U of the equilibrated matrix A (see the description of A for the form of the equilibrated matrix).

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains the pivot indices from the factorization A = P*L*U as computed by DGETRF; row i of the matrix was interchanged with row IPIV(i).

If FACT = 'N', then IPIV is an output argument and on exit contains the pivot indices from the factorization A = P*L*U of the original matrix A.

If FACT = 'E', then IPIV is an output argument and on exit contains the pivot indices from the factorization A = P*L*U of the equilibrated matrix A.

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'R': Row equilibration, i.e., A has been premultiplied by diag(R). = 'C': Column equilibration, i.e., A has been postmultiplied by diag(C). = 'B': Both row and column equilibration, i.e., A has been replaced by diag(R) * A * diag(C). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

R Input and output parameter.

R is DOUBLE PRECISION

R is an array, dimension (N). The row scale factors for A. If EQUED = 'R' or 'B', A is multiplied on the left by diag(R); if EQUED = 'N' or 'C', R is not accessed. R is an input argument if FACT = 'F'; otherwise, R is an output argument. If FACT = 'F' and EQUED = 'R' or 'B', each element of R must be positive.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). The column scale factors for A. If EQUED = 'C' or 'B', A is multiplied on the right by diag(C); if EQUED = 'N' or 'R', C is not accessed. C is an input argument if FACT = 'F'; otherwise, C is an output argument. If FACT = 'F' and EQUED = 'C' or 'B', each element of C must be positive.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if EQUED = 'N', B is not modified; if TRANS = 'N' and EQUED = 'R' or 'B', B is overwritten by diag(R)*B; if TRANS = 'T' or 'C' and EQUED = 'C' or 'B', B is overwritten by diag(C)*B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X to the original system of equations. Note that A and B are modified on exit if EQUED .ne. 'N', and the solution to the equilibrated system is inv(diag(C))* X if TRANS = 'N' and EQUED = 'C' or 'B', or inv(diag(R))* X if TRANS = 'T' or 'C' and EQUED = 'R' or 'B'.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The estimate of the reciprocal condition number of the matrix A after equilibration (if done). If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (4*N). On exit, WORK(1) contains the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If WORK(1) is much less than 1, then the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, condition estimator RCOND, and forward error bound FERR could be unreliable. If factorization fails with 0<INFO<=N, then WORK(1) contains the reciprocal pivot growth factor for the leading INFO columns of A.

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+1: U is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *cgesvx*, *sgesvx* and *zgesvx*. It also exists with a native C interface as *LAPACKE_dgesvx*.

4.6.61 dgesvxx

```
DGESVXX uses the LU factorization to compute the solution to a
   double precision system of linear equations  A * X = B,  where A is an
   N-by-N matrix and X and B are N-by-NRHS matrices.

   If requested, both normwise and maximum componentwise error bounds
   are returned. DGESVXX will return a solution with a tiny
   guaranteed error (O(eps) where eps is the working machine
   precision) unless the matrix is very ill-conditioned, in which
   case a warning is returned. Relevant condition numbers also are
   calculated and returned.

   DGESVXX accepts user-provided factorizations and equilibration
   factors; see the definitions of the FACT and EQUED options.
   Solving with refinement and using a factorization from a previous
   DGESVXX call will also produce a solution with either O(eps)
   errors or warnings, but we cannot make that claim for general
   user-provided factorizations and equilibration factors if they
   differ from what DGESVXX would itself produce.

                                                                    The␣
↪following steps are performed:

   1. If FACT = 'E', double precision scaling factors are computed to equilibrate
   the system:

     TRANS = 'N':  diag(R)*A*diag(C)     *inv(diag(C))*X = diag(R)*B
     TRANS = 'T': (diag(R)*A*diag(C))**T *inv(diag(R))*X = diag(C)*B
     TRANS = 'C': (diag(R)*A*diag(C))**H *inv(diag(R))*X = diag(C)*B

   Whether or not the system will be equilibrated depends on the
   scaling of the matrix A, but if equilibration is used, A is
   overwritten by diag(R)*A*diag(C) and B by diag(R)*B (if TRANS='N')
   or diag(C)*B (if TRANS = 'T' or 'C').

   2. If FACT = 'N' or 'E', the LU decomposition is used to factor
   the matrix A (after equilibration if FACT = 'E') as

     A = P * L * U,

   where P is a permutation matrix, L is a unit lower triangular
   matrix, and U is upper triangular.

   3. If some U(i,i)=0, so that U is exactly singular, then the
   routine returns with INFO = i. Otherwise, the factored form of A
   is used to estimate the condition number of the matrix A (see
   argument RCOND). If the reciprocal of the condition number is less
   than machine precision, the routine still goes on to solve for X
   and compute error bounds as described below.

   4. The system of equations is solved for X using the factored form
   of A.

   5. By default (unless PARAMS(LA_LINRX_ITREF_I) is set to zero),
   the routine will use iterative refinement to try to get a small
   error and error bounds.  Refinement calculates the residual to at
   least twice the working precision.

   6. If equilibration was used, the matrix X is premultiplied by
   diag(C) (if TRANS = 'N') or diag(R) (if TRANS = 'T' or 'C') so
   that it solves the original system before equilibration.
↪   Some optional parameters are bundled in the PARAMS array.  These
    settings determine how refinement is performed, but often the
```

(continues on next page)

(continued from previous page)

```
    defaults are acceptable.  If the defaults are acceptable, users
    can pass NPARAMS = 0 which prevents the source code from accessing
    the PARAMS argument.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dgesvxx(FACT, TRANS, N, NRHS, A, LDA, AF, LDAF, IPIV, EQUED, R, C,
                   B, LDB, X, LDX, RCOND, RPVGRW, BERR, N_ERR_BNDS,
                   ERR_BNDS_NORM, ERR_BNDS_COMP, NPARAMS, PARAMS, WORK, IWORK,
                   INFO)
```

C specification:

```
#include "armpl.h"

void dgesvxx_(const char *fact, const char *trans, const armpl_int_t *n,
              const armpl_int_t *nrhs, double *a, const armpl_int_t *lda,
              double *af, const armpl_int_t *ldaf, armpl_int_t *ipiv,
              char *equed, double *r, double *c, double *b,
              const armpl_int_t *ldb, double *x, const armpl_int_t *ldx,
              double *rcond, double *rpvgrw, double *berr,
              const armpl_int_t *n_err_bnds, double *err_bnds_norm,
              double *err_bnds_comp, const armpl_int_t *nparams,
              double *params, double *work, armpl_int_t *iwork,
              armpl_int_t *info, ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AF and IPIV contain the factored form of A. If EQUED is not 'N', the matrix A has been equilibrated with scaling factors given by R and C. A, AF, and IPIV are not modified. = 'N': The matrix A will be copied to AF and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AF and factored.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate Transpose = Transpose)

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A. If FACT = 'F' and EQUED is not 'N', then A must have been equilibrated by the scaling factors in R and/or C. A is not modified if FACT = 'F' or 'N', or if FACT = 'E' and EQUED = 'N' on exit.

On exit, if EQUED .ne. 'N', A is scaled as follows: EQUED = 'R': A := diag(R) * A EQUED = 'C': A := A * diag(C) EQUED = 'B': A := diag(R) * A * diag(C).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input and output parameter.

AF is DOUBLE PRECISION

AF is an array, dimension (LDAF, N). If FACT = 'F', then AF is an input argument and on entry contains the factors L and U from the factorization A = P*L*U as computed by DGETRF. If EQUED .ne. 'N', then AF is the factored form of the equilibrated matrix A.

If FACT = 'N', then AF is an output argument and on exit returns the factors L and U from the factorization A = P*L*U of the original matrix A.

If FACT = 'E', then AF is an output argument and on exit returns the factors L and U from the factorization A = P*L*U of the equilibrated matrix A (see the description of A for the form of the equilibrated matrix).

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains the pivot indices from the factorization A = P*L*U as computed by DGETRF; row i of the matrix was interchanged with row IPIV(i).

If FACT = 'N', then IPIV is an output argument and on exit contains the pivot indices from the factorization A = P*L*U of the original matrix A.

If FACT = 'E', then IPIV is an output argument and on exit contains the pivot indices from the factorization A = P*L*U of the equilibrated matrix A.

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'R': Row equilibration, i.e., A has been premultiplied by diag(R). = 'C': Column equilibration, i.e., A has been postmultiplied by diag(C). = 'B': Both row and column equilibration, i.e., A has been replaced by diag(R) * A * diag(C). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

R Input and output parameter.

R is DOUBLE PRECISION

R is an array, dimension (N). The row scale factors for A. If EQUED = 'R' or 'B', A is multiplied on the left by diag(R); if EQUED = 'N' or 'C', R is not accessed. R is an input argument if FACT = 'F'; otherwise, R is an output argument. If FACT = 'F' and EQUED = 'R' or 'B', each element of R must be positive. If R is output, each element of R is a power of the radix. If R is input, each element of R should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). The column scale factors for A. If EQUED = 'C' or 'B', A is multiplied on the right by diag(C); if EQUED = 'N' or 'R', C is not accessed. C is an input argument if FACT = 'F'; otherwise, C is an output argument. If FACT = 'F' and EQUED = 'C' or 'B', each element of C must be positive. If C is output, each element of C is a power of the radix. If C is input, each element of C should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if EQUED = 'N', B is not modified; if TRANS = 'N' and EQUED = 'R' or 'B', B is overwritten by diag(R)*B; if TRANS = 'T' or 'C' and EQUED = 'C' or 'B', B is overwritten by diag(C)*B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, NRHS). If INFO = 0, the N-by-NRHS solution matrix X to the original system of equations. Note that A and B are modified on exit if EQUED .ne. 'N', and the solution to the equilibrated system is inv(diag(C))* X if TRANS = 'N' and EQUED = 'C' or 'B', or inv(diag(R))* X if TRANS = 'T' or 'C' and EQUED = 'R' or 'B'.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

RPVGRW Output parameter.

RPVGRW is DOUBLE PRECISION

Reciprocal pivot growth. On exit, this contains the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If this is much less than 1, then the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, estimated condition numbers, and error bounds could be unreliable. If factorization fails with 0<INFO<=N, then this contains the reciprocal pivot growth factor for the leading INFO columns of A. In DGESVX, this quantity is returned in WORK(1).

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). Componentwise relative backward error. This is the componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

N_ERR_BNDS Input parameter.

N_ERR_BNDS is INTEGER

Number of error bounds to return for each right hand side and each type (normwise or componentwise). See ERR_BNDS_NORM and ERR_BNDS_COMP below.

ERR_BNDS_NORM Output parameter.

ERR_BNDS_NORM is DOUBLE PRECISION

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * dlamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * dlamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * dlamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Output parameter.

ERR_BNDS_COMP is DOUBLE PRECISION

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:, N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * dlamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * dlamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * dlamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix

Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

NPARAMS Input parameter.

NPARAMS is INTEGER

Specifies the number of parameters set in PARAMS. If .LE. 0, the PARAMS array is never referenced and default values are used.

PARAMS Input and output parameter.

PARAMS is DOUBLE PRECISION

PARAMS is an array, dimension (NPARAMS). Specifies algorithm parameters. If an entry is .LT. 0.0, then that entry will be filled with default value used for that parameter. Only positions up to NPARAMS are accessed; defaults are used for higher-numbered parameters.

PARAMS(LA_LINRX_ITREF_I = 1) : Whether to perform iterative refinement or not. Default: 1.0D+0 = 0.0 : No refinement is performed, and no error bounds are computed. = 1.0 : Use the extra-precise refinement algorithm. (other values are reserved for future use)

PARAMS(LA_LINRX_ITHRESH_I = 2) : Maximum number of residual computations allowed for refinement. Default: 10 Aggressive: Set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in err_bnds_norm and err_bnds_comp may no longer be trustworthy.

PARAMS(LA_LINRX_CWISE_I = 3) : Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Default: 1.0 (attempt componentwise convergence)

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (4*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. The solution to every right-hand side is guaranteed. < 0: If INFO = -i, the i-th argument had an illegal value > 0 and <= N: U(INFO, INFO) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+J: The solution corresponding to the Jth right-hand side is not guaranteed. The solutions corresponding to other right- hand sides K with K > J may not be guaranteed as well, but only the first such right-hand side is reported. If a small componentwise error is not requested (PARAMS(3) = 0.0) then the Jth right-hand side is the first with a normwise error bound that is not guaranteed (the smallest J such that ERR_BNDS_NORM(J,1) = 0.0). By default (PARAMS(3) = 1.0) the Jth right-hand side is the first with either a normwise or componentwise error bound that is not guaranteed (the smallest J such that either ERR_BNDS_NORM(J,1) = 0.0 or ERR_BNDS_COMP(J,1) = 0.0). See the definition of ERR_BNDS_NORM(:,1) and ERR_BNDS_COMP(:,1). To get information about all of the right-hand sides check ERR_BNDS_NORM or ERR_BNDS_COMP.

Related Information

For this routine in other precisions, please see *cgesvxx*, *sgesvxx* and *zgesvxx*. It also exists with a native C interface as *LAPACKE_dgesvxx*.

4.6.62 dgetrs

dgetrs solves a system of linear equations

```
A * X = B  or  A**T * X = B
```

with a general N-by-N matrix A using the LU factorization computed by DGETRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgetrs(TRANS, N, NRHS, A, LDA, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void dgetrs_(const char *trans, const armpl_int_t *n, const armpl_int_t *nrhs,
             const double *a, const armpl_int_t *lda, const armpl_int_t *ipiv,
             double *b, const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^T * X = B (Conjugate transpose = Transpose)

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). The factors L and U from the factorization A = P*L*U as computed by DGETRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from DGETRF; for 1<=i<=N, row i of the matrix was interchanged with row IPIV(i).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgetrs*, *sgetrs* and *zgetrs*. It also exists with a native C interface as *LAPACKE_dgetrs*.

4.6.63 dgtsv

dgtsv solves the equation

```
A*X = B,
```

where A is an n by n tridiagonal matrix, by Gaussian elimination with partial pivoting.

Note that the equation A^T *X = B may be solved by interchanging the order of the arguments DU and DL.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgtsv(N, NRHS, DL, D, DU, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void dgtsv_(const armpl_int_t *n, const armpl_int_t *nrhs, double *dl,
            double *d, double *du, double *b, const armpl_int_t *ldb,
            armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

DL Input and output parameter.

DL is DOUBLE PRECISION

DL is an array, dimension (N-1). On entry, DL must contain the (n-1) sub-diagonal elements of A.

On exit, DL is overwritten by the (n-2) elements of the second super-diagonal of the upper triangular matrix U from the LU factorization of A, in DL(1), . . . , DL(n-2).

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry, D must contain the diagonal elements of A.

On exit, D is overwritten by the n diagonal elements of U.

DU Input and output parameter.

DU is DOUBLE PRECISION

DU is an array, dimension (N-1). On entry, DU must contain the (n-1) super-diagonal elements of A.

On exit, DU is overwritten by the (n-1) elements of the first super-diagonal of U.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the N by NRHS matrix of right hand side matrix B. On exit, if INFO = 0, the N by NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, U(i,i) is exactly zero, and the solution has not been computed. The factorization has not been completed unless i = N.

Related Information

For this routine in other precisions, please see *cgtsv*, *sgtsv* and *zgtsv*. It also exists with a native C interface as *LAPACKE_dgtsv*.

4.6.64 dgtsvx

dgtsvx uses the LU factorization to compute the solution to a real system of linear equations A * X = B or A^T * X = B, where A is a tridiagonal matrix of order N and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed:
```

1. If FACT = 'N', the LU decomposition is used to factor the matrix A

```
 as A = L * U, where L is a product of permutation and unit lower
bidiagonal matrices and U is upper triangular with nonzeros in
only the main diagonal and first two superdiagonals.
```

2. If some U(i,i)=0, so that U is exactly singular, then the routine

```
 returns with INFO = i. Otherwise, the factored form of A is used
to estimate the condition number of the matrix A.  If the
reciprocal of the condition number is less than machine precision,
INFO = N+1 is returned as a warning, but the routine still goes on
to solve for X and compute error bounds as described below.
```

3. The system of equations is solved for X using the factored form

```
of A.
```

4. Iterative refinement is applied to improve the computed solution

```
 matrix and calculate error bounds and backward error estimates
for it.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dgtsvx(FACT, TRANS, N, NRHS, DL, D, DU, DLF, DF, DUF, DU2, IPIV, B,
                  LDB, X, LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgtsvx_(const char *fact, const char *trans, const armpl_int_t *n,
             const armpl_int_t *nrhs, const double *dl, const double *d,
             const double *du, double *dlf, double *df, double *duf,
             double *du2, armpl_int_t *ipiv, const double *b,
             const armpl_int_t *ldb, double *x, const armpl_int_t *ldx,
             double *rcond, double *ferr, double *berr, double *work,
             armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of A has been supplied on entry. = 'F': DLF, DF, DUF, DU2, and IPIV contain the factored form of A; DL, D, DU, DLF, DF, DUF, DU2 and IPIV will not be modified. = 'N': The matrix will be copied to DLF, DF, and DUF and factored.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose = Transpose)

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

DL Input parameter.

DL is DOUBLE PRECISION

DL is an array, dimension (N-1). The (n-1) subdiagonal elements of A.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The n diagonal elements of A.

DU Input parameter.

DU is DOUBLE PRECISION

DU is an array, dimension (N-1). The (n-1) superdiagonal elements of A.

DLF Input and output parameter.

DLF is DOUBLE PRECISION

DLF is an array, dimension (N-1). If FACT = 'F', then DLF is an input argument and on entry contains the (n-1) multipliers that define the matrix L from the LU factorization of A as computed by DGTTRF.

If FACT = 'N', then DLF is an output argument and on exit contains the (n-1) multipliers that define the matrix L from the LU factorization of A.

DF Input and output parameter.

DF is DOUBLE PRECISION

DF is an array, dimension (N). If FACT = 'F', then DF is an input argument and on entry contains the n diagonal elements of the upper triangular matrix U from the LU factorization of A.

If FACT = 'N', then DF is an output argument and on exit contains the n diagonal elements of the upper triangular matrix U from the LU factorization of A.

DUF Input and output parameter.

DUF is DOUBLE PRECISION

DUF is an array, dimension (N-1). If FACT = 'F', then DUF is an input argument and on entry contains the (n-1) elements of the first superdiagonal of U.

If FACT = 'N', then DUF is an output argument and on exit contains the (n-1) elements of the first superdiagonal of U.

DU2 Input and output parameter.

DU2 is DOUBLE PRECISION

DU2 is an array, dimension (N-2). If FACT = 'F', then DU2 is an input argument and on entry contains the (n-2) elements of the second superdiagonal of U.

If FACT = 'N', then DU2 is an output argument and on exit contains the (n-2) elements of the second superdiagonal of U.

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains the pivot indices from the LU factorization of A as computed by DGTTRF.

If FACT = 'N', then IPIV is an output argument and on exit contains the pivot indices from the LU factorization of A; row i of the matrix was interchanged with row IPIV(i). IPIV(i) will always be either i or i+1; IPIV(i) = i indicates a row interchange was not required.

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). The N-by-NRHS right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The estimate of the reciprocal condition number of the matrix A. If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: U(i,i) is exactly zero. The factorization has not been completed unless i = N, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+1: U is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *cgtsvx*, *sgtsvx* and *zgtsvx*. It also exists with a native C interface as *LAPACKE_dgtsvx*.

4.6.65 dgttrs

dgttrs solves one of the systems of equations

```
A*X = B  or  A**T*X = B,
```

with a tridiagonal matrix A using the LU factorization computed by DGTTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgttrs(TRANS, N, NRHS, DL, D, DU, DU2, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void dgttrs_(const char *trans, const armpl_int_t *n, const armpl_int_t *nrhs,
             const double *dl, const double *d, const double *du,
             const double *du2, const armpl_int_t *ipiv, double *b,
             const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations. = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^T * X = B (Conjugate transpose = Transpose)

N Input parameter.

N is INTEGER

The order of the matrix A.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

DL Input parameter.

DL is DOUBLE PRECISION

DL is an array, dimension (N-1). The (n-1) multipliers that define the matrix L from the LU factorization of A.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The n diagonal elements of the upper triangular matrix U from the LU factorization of A.

DU Input parameter.

DU is DOUBLE PRECISION

DU is an array, dimension (N-1). The (n-1) elements of the first super-diagonal of U.

DU2 Input parameter.

DU2 is DOUBLE PRECISION

DU2 is an array, dimension (N-2). The (n-2) elements of the second super-diagonal of U.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices; for 1 <= i <= n, row i of the matrix was interchanged with row IPIV(i). IPIV(i) will always be either i or i+1; IPIV(i) = i indicates a row interchange was not required.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the matrix of right hand side vectors B. On exit, B is overwritten by the solution vectors X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgttrs*, *sgttrs* and *zgttrs*. It also exists with a native C interface as *LAPACKE_dgttrs*.

4.6.66 dpbsv

dpbsv computes the solution to a real system of linear equations

```
A * X = B,
```

where A is an N-by-N symmetric positive definite band matrix and X and B are N-by-NRHS matrices.

The Cholesky decomposition is used to factor A as

```
A = U**T * U,  if UPLO = 'U', or
A = L * L**T,  if UPLO = 'L',
```

where U is an upper triangular band matrix, and L is a lower triangular band matrix, with the same number of superdiagonals or subdiagonals as A. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine dpbsv(UPLO, N, KD, NRHS, AB, LDAB, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void dpbsv_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
            const armpl_int_t *nrhs, double *ab, const armpl_int_t *ldab,
            double *b, const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AB Input and output parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(KD+1+i-j,j) = A(i,j) for max(1,j-KD)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(N,j+KD). See below for further details.

On exit, if INFO = 0, the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T of the band matrix A, in the same storage format as A.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

Related Information

For this routine in other precisions, please see *cpbsv*, *spbsv* and *zpbsv*. It also exists with a native C interface as *LAPACKE_dpbsv*.

4.6.67 dpbsvx

dpbsvx uses the Cholesky factorization A = U^T *U or A = L*L^T to compute the solution to a real system of linear equations

```
A * X = B,
```

where A is an N-by-N symmetric positive definite band matrix and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed:
```

1. If FACT = 'E', real scaling factors are computed to equilibrate

```
 the system:
   diag(S) * A * diag(S) * inv(diag(S)) * X = diag(S) * B
Whether or not the system will be equilibrated depends on the
scaling of the matrix A, but if equilibration is used, A is
overwritten by diag(S)*A*diag(S) and B by diag(S)*B.
```

2. If FACT = 'N' or 'E', the Cholesky decomposition is used to

```
 factor the matrix A (after equilibration if FACT = 'E') as
   A = U**T * U,  if UPLO = 'U', or
   A = L * L**T,  if UPLO = 'L',
where U is an upper triangular band matrix, and L is a lower
triangular band matrix.
```

3. If the leading i-by-i principal minor is not positive definite,

```
 then the routine returns with INFO = i. Otherwise, the factored
form of A is used to estimate the condition number of the matrix
A.  If the reciprocal of the condition number is less than machine
precision, INFO = N+1 is returned as a warning, but the routine
still goes on to solve for X and compute error bounds as
described below.
```

4. The system of equations is solved for X using the factored form

```
of A.
```

5. Iterative refinement is applied to improve the computed solution

```
 matrix and calculate error bounds and backward error estimates
for it.
```

6. If equilibration was used, the matrix X is premultiplied by

```
 diag(S) so that it solves the original system before
equilibration.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dpbsvx(FACT, UPLO, N, KD, NRHS, AB, LDAB, AFB, LDAFB, EQUED, S, B,
                  LDB, X, LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dpbsvx_(const char *fact, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *kd, const armpl_int_t *nrhs, double *ab,
             const armpl_int_t *ldab, double *afb, const armpl_int_t *ldafb,
             char *equed, double *s, double *b, const armpl_int_t *ldb,
             double *x, const armpl_int_t *ldx, double *rcond, double *ferr,
             double *berr, double *work, armpl_int_t *iwork,
             armpl_int_t *info, ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AFB contains the factored form of A. If EQUED = 'Y', the matrix A has been equilibrated with scaling factors given by S. AB and AFB will not be modified. = 'N': The matrix A will be copied to AFB and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AFB and factored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right-hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AB Input and output parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first KD+1 rows of the array, except if FACT = 'F' and EQUED = 'Y', then A must contain the equilibrated matrix diag(S)*A*diag(S). The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(KD+1+i-j,j) = A(i,j) for max(1,j-KD)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(N,j+KD). See below for further details.

On exit, if FACT = 'E' and EQUED = 'Y', A is overwritten by diag(S)*A*diag(S).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array A. LDAB >= KD+1.

AFB Input and output parameter.

AFB is DOUBLE PRECISION

AFB is an array, dimension (LDAFB, N). If FACT = 'F', then AFB is an input argument and on entry contains the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T of the band matrix A, in the same storage format as A (see AB). If EQUED = 'Y', then AFB is the factored form of the equilibrated matrix A.

If FACT = 'N', then AFB is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T .

If FACT = 'E', then AFB is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T of the equilibrated matrix A (see the description of A for the form of the equilibrated matrix).

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AFB. LDAFB >= KD+1.

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'Y': Equilibration was done, i.e., A has been replaced by diag(S) * A * diag(S). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

S Input and output parameter.

S is DOUBLE PRECISION

S is an array, dimension (N). The scale factors for A; not accessed if EQUED = 'N'. S is an input argument if FACT = 'F'; otherwise, S is an output argument. If FACT = 'F' and EQUED = 'Y', each element of S must be positive.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if EQUED = 'N', B is not modified; if EQUED = 'Y', B is overwritten by diag(S) * B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X to the original system of equations. Note that if EQUED = 'Y', A and B are modified on exit, and the solution to the equilibrated system is inv(diag(S))*X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The estimate of the reciprocal condition number of the matrix A after equilibration (if done). If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed. RCOND = 0 is returned. = N+1: U is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *cpbsvx*, *spbsvx* and *zpbsvx*. It also exists with a native C interface as *LAPACKE_dpbsvx*.

4.6.68 dpbtrs

dpbtrs solves a system of linear equations A*X = B with a symmetric positive definite band matrix A using the Cholesky factorization $A = U^T * U$ or $A = L * L^T$ computed by DPBTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine dpbtrs(UPLO, N, KD, NRHS, AB, LDAB, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void dpbtrs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
             const armpl_int_t *nrhs, const double *ab,
             const armpl_int_t *ldab, double *b, const armpl_int_t *ldb,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangular factor stored in AB; = 'L': Lower triangular factor stored in AB.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AB Input parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). The triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T of the band matrix A, stored in the first KD+1 rows of the array. The j-th column of U or L is stored in the j-th column of the array AB as follows: if UPLO ='U', AB(kd+1+i-j,j) = U(i,j) for max(1,j-kd)<=i<=j; if UPLO ='L', AB(1+i-j,j) = L(i,j) for j<=i<=min(n,j+kd).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cpbtrs*, *spbtrs* and *zpbtrs*. It also exists with a native C interface as *LAPACKE_dpbtrs*.

4.6.69 dpftrs

dpftrs solves a system of linear equations A*X = B with a symmetric positive definite matrix A using the Cholesky factorization A = U^T *U or A = L*L^T computed by DPFTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine dpftrs(TRANSR, UPLO, N, NRHS, A, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void dpftrs_(const char *transr, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *nrhs, const double *a, double *b,
             const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': The Normal TRANSR of RFP A is stored; = 'T': The Transpose TRANSR of RFP A is stored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of RFP A is stored; = 'L': Lower triangle of RFP A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (N*(N+1)/2).. The triangular factor U or L from the Cholesky factorization of RFP A = U^T *U or RFP A = L*L^T , as computed by DPFTRF. See note below for more details about RFP A.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cpftrs*, *spftrs* and *zpftrs*. It also exists with a native C interface as *LAPACKE_dpftrs*.

4.6.70 dposv

dposv computes the solution to a real system of linear equations

```
A * X = B,
```

where A is an N-by-N symmetric positive definite matrix and X and B are N-by-NRHS matrices.

The Cholesky decomposition is used to factor A as

```
A = U**T* U,  if UPLO = 'U', or
A = L * L**T,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is a lower triangular matrix. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine dposv(UPLO, N, NRHS, A, LDA, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void dposv_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
            double *a, const armpl_int_t *lda, double *b,
            const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T .

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

Related Information

For this routine in other precisions, please see *cposv*, *sposv* and *zposv*. It also exists with a native C interface as *LAPACKE_dposv*.

4.6.71 dposvx

dposvx uses the Cholesky factorization A = U^T *U or A = L*L^T to compute the solution to a real system of linear equations

```
A * X = B,
```

where A is an N-by-N symmetric positive definite matrix and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed:
```

1. If FACT = 'E', real scaling factors are computed to equilibrate

```
 the system:
   diag(S) * A * diag(S) * inv(diag(S)) * X = diag(S) * B
Whether or not the system will be equilibrated depends on the
scaling of the matrix A, but if equilibration is used, A is
overwritten by diag(S)*A*diag(S) and B by diag(S)*B.
```

2. If FACT = 'N' or 'E', the Cholesky decomposition is used to

```
 factor the matrix A (after equilibration if FACT = 'E') as
   A = U**T* U,  if UPLO = 'U', or
   A = L * L**T,  if UPLO = 'L',
where U is an upper triangular matrix and L is a lower triangular
matrix.
```

3. If the leading i-by-i principal minor is not positive definite,

```
 then the routine returns with INFO = i. Otherwise, the factored
form of A is used to estimate the condition number of the matrix
A.  If the reciprocal of the condition number is less than machine
precision, INFO = N+1 is returned as a warning, but the routine
still goes on to solve for X and compute error bounds as
described below.
```

4. The system of equations is solved for X using the factored form

```
of A.
```

5. Iterative refinement is applied to improve the computed solution

```
 matrix and calculate error bounds and backward error estimates
for it.
```

6. If equilibration was used, the matrix X is premultiplied by

```
 diag(S) so that it solves the original system before
equilibration.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dposvx(FACT, UPLO, N, NRHS, A, LDA, AF, LDAF, EQUED, S, B, LDB, X,
                  LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dposvx_(const char *fact, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *nrhs, double *a, const armpl_int_t *lda,
             double *af, const armpl_int_t *ldaf, char *equed, double *s,
             double *b, const armpl_int_t *ldb, double *x,
             const armpl_int_t *ldx, double *rcond, double *ferr,
             double *berr, double *work, armpl_int_t *iwork,
             armpl_int_t *info, ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AF contains the factored form of A. If EQUED = 'Y', the matrix A has been equilibrated with scaling factors given by S. A and AF will not be modified. = 'N': The matrix A will be copied to AF and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AF and factored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A, except if FACT = 'F' and EQUED = 'Y', then A must contain the equilibrated matrix diag(S)*A*diag(S). If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. A is not modified if FACT = 'F' or 'N', or if FACT = 'E' and EQUED = 'N' on exit.

On exit, if FACT = 'E' and EQUED = 'Y', A is overwritten by diag(S)*A*diag(S).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input and output parameter.

AF is DOUBLE PRECISION

AF is an array, dimension (LDAF, N). If FACT = 'F', then AF is an input argument and on entry contains the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , in the same storage format as A. If EQUED .ne. 'N', then AF is the factored form of the equilibrated matrix diag(S)*A*diag(S).

If FACT = 'N', then AF is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T of the original matrix A.

If FACT = 'E', then AF is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T of the equilibrated matrix A (see the description of A for the form of the equilibrated matrix).

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'Y': Equilibration was done, i.e., A has been replaced by diag(S) * A * diag(S). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

S Input and output parameter.

S is DOUBLE PRECISION

S is an array, dimension (N). The scale factors for A; not accessed if EQUED = 'N'. S is an input argument if FACT = 'F'; otherwise, S is an output argument. If FACT = 'F' and EQUED = 'Y', each element of S must be positive.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if EQUED = 'N', B is not modified; if EQUED = 'Y', B is overwritten by diag(S) * B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X to the original system of equations. Note that if EQUED = 'Y', A and B are modified on exit, and the solution to the equilibrated system is inv(diag(S))*X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The estimate of the reciprocal condition number of the matrix A after equilibration (if done). If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed. RCOND = 0 is returned. = N+1: U is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *cposvx*, *sposvx* and *zposvx*. It also exists with a native C interface as *LAPACKE_dposvx*.

4.6.72 dposvxx

```
DPOSVXX uses the Cholesky factorization A = U**T*U or A = L*L**T
   to compute the solution to a double precision system of linear equations
   A * X = B, where A is an N-by-N symmetric positive definite matrix
   and X and B are N-by-NRHS matrices.

   If requested, both normwise and maximum componentwise error bounds
   are returned. DPOSVXX will return a solution with a tiny
   guaranteed error (O(eps) where eps is the working machine
   precision) unless the matrix is very ill-conditioned, in which
   case a warning is returned. Relevant condition numbers also are
   calculated and returned.

   DPOSVXX accepts user-provided factorizations and equilibration
   factors; see the definitions of the FACT and EQUED options.
   Solving with refinement and using a factorization from a previous
   DPOSVXX call will also produce a solution with either O(eps)
   errors or warnings, but we cannot make that claim for general
   user-provided factorizations and equilibration factors if they
   differ from what DPOSVXX would itself produce.

                                                                  The␣
↪following steps are performed:

   1. If FACT = 'E', double precision scaling factors are computed to equilibrate
   the system:

     diag(S)*A*diag(S)      *inv(diag(S))*X = diag(S)*B

   Whether or not the system will be equilibrated depends on the
   scaling of the matrix A, but if equilibration is used, A is
   overwritten by diag(S)*A*diag(S) and B by diag(S)*B.

   2. If FACT = 'N' or 'E', the Cholesky decomposition is used to
   factor the matrix A (after equilibration if FACT = 'E') as
      A = U**T* U,  if UPLO = 'U', or
      A = L * L**T,  if UPLO = 'L',
   where U is an upper triangular matrix and L is a lower triangular
   matrix.
```

(continues on next page)

(continued from previous page)

```
   3. If the leading i-by-i principal minor is not positive definite,
   then the routine returns with INFO = i. Otherwise, the factored
   form of A is used to estimate the condition number of the matrix
   A (see argument RCOND).  If the reciprocal of the condition number
   is less than machine precision, the routine still goes on to solve
   for X and compute error bounds as described below.

   4. The system of equations is solved for X using the factored form
   of A.

   5. By default (unless PARAMS(LA_LINRX_ITREF_I) is set to zero),
   the routine will use iterative refinement to try to get a small
   error and error bounds.  Refinement calculates the residual to at
   least twice the working precision.

   6. If equilibration was used, the matrix X is premultiplied by
   diag(S) so that it solves the original system before
   equilibration.                          Some optional parameters are bundled␣
↪in the PARAMS array.  These
    settings determine how refinement is performed, but often the
    defaults are acceptable.  If the defaults are acceptable, users
    can pass NPARAMS = 0 which prevents the source code from accessing
    the PARAMS argument.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dposvxx(FACT, UPLO, N, NRHS, A, LDA, AF, LDAF, EQUED, S, B, LDB, X,
                   LDX, RCOND, RPVGRW, BERR, N_ERR_BNDS, ERR_BNDS_NORM,
                   ERR_BNDS_COMP, NPARAMS, PARAMS, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dposvxx_(const char *fact, const char *uplo, const armpl_int_t *n,
              const armpl_int_t *nrhs, double *a, const armpl_int_t *lda,
              double *af, const armpl_int_t *ldaf, char *equed, double *s,
              double *b, const armpl_int_t *ldb, double *x,
              const armpl_int_t *ldx, double *rcond, double *rpvgrw,
              double *berr, const armpl_int_t *n_err_bnds,
              double *err_bnds_norm, double *err_bnds_comp,
              const armpl_int_t *nparams, double *params, double *work,
              armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AF contains the factored form of A. If EQUED is not 'N', the matrix A has been equilibrated with scaling factors given by S. A and AF are not modified. = 'N': The matrix A will be copied to AF and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AF and factored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A, except if FACT = 'F' and EQUED = 'Y', then A must contain the equilibrated matrix diag(S)*A*diag(S). If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. A is not modified if FACT = 'F' or 'N', or if FACT = 'E' and EQUED = 'N' on exit.

On exit, if FACT = 'E' and EQUED = 'Y', A is overwritten by diag(S)*A*diag(S).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input and output parameter.

AF is DOUBLE PRECISION

AF is an array, dimension (LDAF, N). If FACT = 'F', then AF is an input argument and on entry contains the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , in the same storage format as A. If EQUED .ne. 'N', then AF is the factored form of the equilibrated matrix diag(S)*A*diag(S).

If FACT = 'N', then AF is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T of the original matrix A.

If FACT = 'E', then AF is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T of the equilibrated matrix A (see the description of A for the form of the equilibrated matrix).

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'Y': Both row and column equilibration, i.e., A has been replaced by diag(S) * A * diag(S). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

S Input and output parameter.

S is DOUBLE PRECISION

S is an array, dimension (N). The row scale factors for A. If EQUED = 'Y', A is multiplied on the left and right by diag(S). S is an input argument if FACT = 'F'; otherwise, S is an output argument. If FACT = 'F' and EQUED = 'Y', each element of S must be positive. If S is output, each element of S is a power of the

radix. If S is input, each element of S should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if EQUED = 'N', B is not modified; if EQUED = 'Y', B is overwritten by diag(S)*B;

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, NRHS). If INFO = 0, the N-by-NRHS solution matrix X to the original system of equations. Note that A and B are modified on exit if EQUED .ne. 'N', and the solution to the equilibrated system is inv(diag(S))*X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

RPVGRW Output parameter.

RPVGRW is DOUBLE PRECISION

Reciprocal pivot growth. On exit, this contains the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If this is much less than 1, then the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, estimated condition numbers, and error bounds could be unreliable. If factorization fails with 0<INFO<=N, then this contains the reciprocal pivot growth factor for the leading INFO columns of A.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). Componentwise relative backward error. This is the componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

N_ERR_BNDS Input parameter.

N_ERR_BNDS is INTEGER

Number of error bounds to return for each right hand side and each type (normwise or componentwise). See ERR_BNDS_NORM and ERR_BNDS_COMP below.

ERR_BNDS_NORM Output parameter.

ERR_BNDS_NORM is DOUBLE PRECISION

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * dlamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * dlamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * dlamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Output parameter.

ERR_BNDS_COMP is DOUBLE PRECISION

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:, N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * dlamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * dlamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * dlamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

NPARAMS Input parameter.

NPARAMS is INTEGER

Specifies the number of parameters set in PARAMS. If .LE. 0, the PARAMS array is never referenced and default values are used.

PARAMS Input and output parameter.

PARAMS is DOUBLE PRECISION

PARAMS is an array, dimension NPARAMS. Specifies algorithm parameters. If an entry is .LT. 0.0, then that entry will be filled with default value used for that parameter. Only positions up to NPARAMS are accessed; defaults are used for higher-numbered parameters.

PARAMS(LA_LINRX_ITREF_I = 1) : Whether to perform iterative refinement or not. Default: 1.0D+0 = 0.0 : No refinement is performed, and no error bounds are computed. = 1.0 : Use the extra-precise refinement algorithm. (other values are reserved for future use)

PARAMS(LA_LINRX_ITHRESH_I = 2) : Maximum number of residual computations allowed for refinement. Default: 10 Aggressive: Set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in err_bnds_norm and err_bnds_comp may no longer be trustworthy.

PARAMS(LA_LINRX_CWISE_I = 3) : Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Default: 1.0 (attempt componentwise convergence)

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (4*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. The solution to every right-hand side is guaranteed. < 0: If INFO = -i, the i-th argument had an illegal value > 0 and <= N: U(INFO, INFO) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+J: The solution corresponding to the Jth right-hand side is not guaranteed. The solutions corresponding to other right- hand sides K with K > J may not be guaranteed as well, but only the first such right-hand side is reported. If a small componentwise error is not requested (PARAMS(3) = 0.0) then the Jth right-hand side is the first with a normwise error bound that is not guaranteed (the smallest J such that ERR_BNDS_NORM(J,1) = 0.0). By default (PARAMS(3) = 1.0) the Jth right-hand side is the first with either a normwise or componentwise error bound that is not guaranteed (the smallest J such that either ERR_BNDS_NORM(J,1) = 0.0 or ERR_BNDS_COMP(J,1) = 0.0). See the definition of ERR_BNDS_NORM(:,1) and ERR_BNDS_COMP(:,1). To get information about all of the right-hand sides check ERR_BNDS_NORM or ERR_BNDS_COMP.

Related Information

For this routine in other precisions, please see *cposvxx*, *sposvxx* and *zposvxx*. It also exists with a native C interface as *LAPACKE_dposvxx*.

4.6.73 dpotrs

dpotrs solves a system of linear equations A*X = B with a symmetric positive definite matrix A using the Cholesky factorization $A = U^T * U$ or $A = L * L^T$ computed by DPOTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine dpotrs(UPLO, N, NRHS, A, LDA, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void dpotrs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const double *a, const armpl_int_t *lda, double *b,
             const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). The triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , as computed by DPOTRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cpotrs*, *spotrs* and *zpotrs*. It also exists with a native C interface as *LAPACKE_dpotrs*.

4.6.74 dppsv

dppsv computes the solution to a real system of linear equations

```
A * X = B,
```

where A is an N-by-N symmetric positive definite matrix stored in packed format and X and B are N-by-NRHS matrices.

The Cholesky decomposition is used to factor A as

```
A = U**T* U,  if UPLO = 'U', or
A = L * L**T,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is a lower triangular matrix. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine dppsv(UPLO, N, NRHS, AP, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void dppsv_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
            double *ap, double *b, const armpl_int_t *ldb, armpl_int_t *info,
            ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AP Input and output parameter.

AP is DOUBLE PRECISION

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n. See below for further details.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , in the same storage format as A.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

Related Information

For this routine in other precisions, please see *cppsv*, *sppsv* and *zppsv*. It also exists with a native C interface as *LAPACKE_dppsv*.

4.6.75 dppsvx

dppsvx uses the Cholesky factorization A = U^T *U or A = L*L^T to compute the solution to a real system of linear equations

```
A * X = B,
```

where A is an N-by-N symmetric positive definite matrix stored in packed format and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed:
```

1. If FACT = 'E', real scaling factors are computed to equilibrate

```
 the system:
   diag(S) * A * diag(S) * inv(diag(S)) * X = diag(S) * B
Whether or not the system will be equilibrated depends on the
scaling of the matrix A, but if equilibration is used, A is
overwritten by diag(S)*A*diag(S) and B by diag(S)*B.
```

2. If FACT = 'N' or 'E', the Cholesky decomposition is used to

```
 factor the matrix A (after equilibration if FACT = 'E') as
   A = U**T* U,  if UPLO = 'U', or
   A = L * L**T,  if UPLO = 'L',
where U is an upper triangular matrix and L is a lower triangular
matrix.
```

3. If the leading i-by-i principal minor is not positive definite,

```
 then the routine returns with INFO = i. Otherwise, the factored
form of A is used to estimate the condition number of the matrix
A.  If the reciprocal of the condition number is less than machine
precision, INFO = N+1 is returned as a warning, but the routine
```

(continues on next page)

(continued from previous page)

```
still goes on to solve for X and compute error bounds as
described below.
```

4. The system of equations is solved for X using the factored form

```
of A.
```

5. Iterative refinement is applied to improve the computed solution

```
 matrix and calculate error bounds and backward error estimates
for it.
```

6. If equilibration was used, the matrix X is premultiplied by

```
 diag(S) so that it solves the original system before
equilibration.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dppsvx(FACT, UPLO, N, NRHS, AP, AFP, EQUED, S, B, LDB, X, LDX,
                  RCOND, FERR, BERR, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dppsvx_(const char *fact, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *nrhs, double *ap, double *afp, char *equed,
             double *s, double *b, const armpl_int_t *ldb, double *x,
             const armpl_int_t *ldx, double *rcond, double *ferr,
             double *berr, double *work, armpl_int_t *iwork,
             armpl_int_t *info, ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AFP contains the factored form of A. If EQUED = 'Y', the matrix A has been equilibrated with scaling factors given by S. AP and AFP will not be modified. = 'N': The matrix A will be copied to AFP and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AFP and factored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AP Input and output parameter.

AP is DOUBLE PRECISION

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array, except if FACT = 'F' and EQUED = 'Y', then A must contain the equilibrated matrix diag(S)*A*diag(S). The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n. See below for further details. A is not modified if FACT = 'F' or 'N', or if FACT = 'E' and EQUED = 'N' on exit.

On exit, if FACT = 'E' and EQUED = 'Y', A is overwritten by diag(S)*A*diag(S).

AFP Input and output parameter.

AFP is DOUBLE PRECISION

AFP is an array, dimension (N*(N+1)/2). If FACT = 'F', then AFP is an input argument and on entry contains the triangular factor U or L from the Cholesky factorization $A = U^T * U$ or $A = L * L^T$, in the same storage format as A. If EQUED .ne. 'N', then AFP is the factored form of the equilibrated matrix A.

If FACT = 'N', then AFP is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization $A = U^T * U$ or $A = L * L^T$ of the original matrix A.

If FACT = 'E', then AFP is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization $A = U^T * U$ or $A = L * L^T$ of the equilibrated matrix A (see the description of AP for the form of the equilibrated matrix).

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'Y': Equilibration was done, i.e., A has been replaced by diag(S) * A * diag(S). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

S Input and output parameter.

S is DOUBLE PRECISION

S is an array, dimension (N). The scale factors for A; not accessed if EQUED = 'N'. S is an input argument if FACT = 'F'; otherwise, S is an output argument. If FACT = 'F' and EQUED = 'Y', each element of S must be positive.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if EQUED = 'N', B is not modified; if EQUED = 'Y', B is overwritten by diag(S) * B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X to the original system of equations. Note that if EQUED = 'Y', A and B are modified on exit, and the solution to the equilibrated system is inv(diag(S))*X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The estimate of the reciprocal condition number of the matrix A after equilibration (if done). If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed. RCOND = 0 is returned. = N+1: U is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *cppsvx*, *sppsvx* and *zppsvx*. It also exists with a native C interface as *LAPACKE_dppsvx*.

4.6.76 dpptrs

dpptrs solves a system of linear equations A*X = B with a symmetric positive definite matrix A in packed storage using the Cholesky factorization $A = U^T * U$ or $A = L * L^T$ computed by DPPTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine dpptrs(UPLO, N, NRHS, AP, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void dpptrs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const double *ap, double *b, const armpl_int_t *ldb,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AP Input parameter.

AP is DOUBLE PRECISION

AP is an array, dimension (N*(N+1)/2). The triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , packed columnwise in a linear array. The j-th column of U or L is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = U(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = L(i,j) for j<=i<=n.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cpptrs*, *spptrs* and *zpptrs*. It also exists with a native C interface as *LAPACKE_dpptrs*.

4.6.77 dptsv

`dptsv` computes the solution to a real system of linear equations A*X = B, where A is an N-by-N symmetric positive definite tridiagonal matrix, and X and B are N-by-NRHS matrices.

A is factored as A = L*D*L^T , and the factored form of A is then used to solve the system of equations.

Syntax

Fortran specification:

```
use armpl_library

subroutine dptsv(N, NRHS, D, E, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void dptsv_(const armpl_int_t *n, const armpl_int_t *nrhs, double *d,
            double *e, double *b, const armpl_int_t *ldb, armpl_int_t *info);
```

Parameters

N Input parameter.

`N` is `INTEGER`

The order of the matrix A. `N` >= 0.

NRHS Input parameter.

`NRHS` is `INTEGER`

The number of right hand sides, i.e., the number of columns of the matrix `B`. `NRHS` >= 0.

D Input and output parameter.

`D` is `DOUBLE PRECISION`

`D` is an array, dimension (`N`). On entry, the n diagonal elements of the tridiagonal matrix A. On exit, the n diagonal elements of the diagonal matrix `D` from the factorization A = L*D*L^T .

E Input and output parameter.

`E` is `DOUBLE PRECISION`

`E` is an array, dimension (N-1). On entry, the (n-1) subdiagonal elements of the tridiagonal matrix A. On exit, the (n-1) subdiagonal elements of the unit bidiagonal factor L from the L*D*L^T factorization of A. (E can also be regarded as the superdiagonal of the unit bidiagonal factor U from the U^T *D*U factorization of A.)

B Input and output parameter.

`B` is `DOUBLE PRECISION`

`B` is an array, dimension (`LDB`, `NRHS`). On entry, the N-by-NRHS right hand side matrix `B`. On exit, if `INFO` = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

`LDB` is `INTEGER`

The leading dimension of the array `B`. `LDB` >= max(1, `N`).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i is not positive definite, and the solution has not been computed. The factorization has not been completed unless i = N.

Related Information

For this routine in other precisions, please see *cptsv*, *sptsv* and *zptsv*. It also exists with a native C interface as *LAPACKE_dptsv*.

4.6.78 dptsvx

dptsvx uses the factorization A = L*D*L^T to compute the solution to a real system of linear equations A*X = B, where A is an N-by-N symmetric positive definite tridiagonal matrix and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed:
```

1. If FACT = 'N', the matrix A is factored as A = L*D*L^T , where L

```
 is a unit lower bidiagonal matrix and D is diagonal.  The
factorization can also be regarded as having the form
A = U**T*D*U.
```

2. If the leading i-by-i principal minor is not positive definite,

```
 then the routine returns with INFO = i. Otherwise, the factored
form of A is used to estimate the condition number of the matrix
A.  If the reciprocal of the condition number is less than machine
precision, INFO = N+1 is returned as a warning, but the routine
still goes on to solve for X and compute error bounds as
described below.
```

3. The system of equations is solved for X using the factored form

```
of A.
```

4. Iterative refinement is applied to improve the computed solution

```
 matrix and calculate error bounds and backward error estimates
for it.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dptsvx(FACT, N, NRHS, D, E, DF, EF, B, LDB, X, LDX, RCOND, FERR,
                  BERR, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dptsvx_(const char *fact, const armpl_int_t *n, const armpl_int_t *nrhs,
             const double *d, const double *e, double *df, double *ef,
             const double *b, const armpl_int_t *ldb, double *x,
             const armpl_int_t *ldx, double *rcond, double *ferr,
             double *berr, double *work, armpl_int_t *info, ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of A has been supplied on entry. = 'F': On entry, DF and EF contain the factored form of A. D, E, DF, and EF will not be modified. = 'N': The matrix A will be copied to DF and EF and factored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The n diagonal elements of the tridiagonal matrix A.

E Input parameter.

E is DOUBLE PRECISION

E is an array, dimension (N-1). The (n-1) subdiagonal elements of the tridiagonal matrix A.

DF Input and output parameter.

DF is DOUBLE PRECISION

DF is an array, dimension (N). If FACT = 'F', then DF is an input argument and on entry contains the n diagonal elements of the diagonal matrix D from the L*D*L^T factorization of A. If FACT = 'N', then DF is an output argument and on exit contains the n diagonal elements of the diagonal matrix D from the L*D*L^T factorization of A.

EF Input and output parameter.

EF is DOUBLE PRECISION

EF is an array, dimension (N-1). If FACT = 'F', then EF is an input argument and on entry contains the (n-1) subdiagonal elements of the unit bidiagonal factor L from the L*D*L^T factorization of A. If FACT = 'N', then EF is an output argument and on exit contains the (n-1) subdiagonal elements of the unit bidiagonal factor L from the L*D*L^T factorization of A.

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). The N-by-NRHS right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, NRHS). If INFO = 0 of INFO = N+1, the N-by-NRHS solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The reciprocal condition number of the matrix A. If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j).

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed. RCOND = 0 is returned. = N+1: U is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *cptsvx*, *sptsvx* and *zptsvx*. It also exists with a native C interface as *LAPACKE_dptsvx*.

4.6.79 dpttrs

dpttrs solves a tridiagonal system of the form

```
A * X = B
```

using the L*D*L^T factorization of A computed by DPTTRF. D is a diagonal matrix specified in the vector D, L is a unit bidiagonal matrix whose subdiagonal is specified in the vector E, and X and B are N by NRHS matrices.

Syntax

Fortran specification:

```
use armpl_library

subroutine dpttrs(N, NRHS, D, E, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void dpttrs_(const armpl_int_t *n, const armpl_int_t *nrhs, const double *d,
             const double *e, double *b, const armpl_int_t *ldb,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the tridiagonal matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The n diagonal elements of the diagonal matrix D from the L*D*L^T factorization of A.

E Input parameter.

E is DOUBLE PRECISION

E is an array, dimension (N-1). The (n-1) subdiagonal elements of the unit bidiagonal factor L from the L*D*L^T factorization of A. E can also be regarded as the superdiagonal of the unit bidiagonal factor U from the factorization A = U^T *D*U.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the right hand side vectors B for the system of linear equations. On exit, the solution vectors, X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cpttrs*, *spttrs* and *zpttrs*. It also exists with a native C interface as *LAPACKE_dpttrs*.

4.6.80 dsgesv

dsgesv computes the solution to a real system of linear equations

```
A * X = B,
```

where A is an N-by-N matrix and X and B are N-by-NRHS matrices.

dsgesv first attempts to factorize the matrix in SINGLE PRECISION and use this factorization within an iterative refinement procedure to produce a solution with DOUBLE PRECISION normwise backward error quality (see below). If the approach fails the method switches to a DOUBLE PRECISION factorization and solve.

The iterative refinement is not going to be a winning strategy if the ratio SINGLE PRECISION performance over DOUBLE PRECISION performance is too small. A reasonable strategy should take the number of right-hand sides and the size of the matrix into account. This might be done with a call to ILAENV in the future. Up to now, we always try iterative refinement.

The iterative refinement process is stopped if

```
ITER > ITERMAX
```

or for all the RHS we have:

```
RNRM < SQRT(N)*XNRM*ANRM*EPS*BWDMAX
```

where

```
o ITER is the number of the current iteration in the iterative
  refinement process
o RNRM is the infinity-norm of the residual
o XNRM is the infinity-norm of the solution
o ANRM is the infinity-operator-norm of the matrix A
o EPS is the machine epsilon returned by DLAMCH('Epsilon')
```

The value ITERMAX and BWDMAX are fixed to 30 and 1.0D+00 respectively.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsgesv(N, NRHS, A, LDA, IPIV, B, LDB, X, LDX, WORK, SWORK, ITER,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dsgesv_(const armpl_int_t *n, const armpl_int_t *nrhs, double *a,
             const armpl_int_t *lda, armpl_int_t *ipiv, const double *b,
             const armpl_int_t *ldb, double *x, const armpl_int_t *ldx,
             double *work, float *swork, armpl_int_t *iter,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array,. dimension (LDA, N) On entry, the N-by-N coefficient matrix A. On exit, if iterative refinement has been successfully used (INFO.EQ.0 and ITER.GE.0, see description below), then A is unchanged, if double precision factorization has been used (INFO.EQ.0 and ITER.LT.0, see description below), then the array A contains the factors L and U from the factorization A = P*L*U; the unit diagonal elements of L are not stored.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices that define the permutation matrix P; row i of the matrix was interchanged with row IPIV(i). Corresponds either to the single precision factorization (if INFO.EQ.0 and ITER.GE.0) or the double precision factorization (if INFO.EQ.0 and ITER.LT.0).

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). The N-by-NRHS right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, NRHS). If INFO = 0, the N-by-NRHS solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (N, NRHS). This array is used to hold the residual vectors.

SWORK Output parameter.

SWORK is REAL

SWORK is an array, dimension (N*(N+NRHS)). This array is used to use the single precision matrix and the right-hand sides or solutions in single precision.

ITER Output parameter.

ITER is INTEGER

< 0: iterative refinement has failed, double precision factorization has been performed -1 : the routine fell back to full precision for implementation- or machine-specific reasons -2 : narrowing the precision induced an overflow, the routine fell back to full precision -3 : failure of SGETRF -31: stop the iterative refinement after the 30th iterations > 0: iterative refinement has been successfully used. Returns the number of iterations

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, U(i,i) computed in DOUBLE PRECISION is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

Related Information

It also exists with a native C interface as *LAPACKE_dsgesv*.

4.6.81 dsposv

dsposv computes the solution to a real system of linear equations

```
A * X = B,
```

where A is an N-by-N symmetric positive definite matrix and X and B are N-by-NRHS matrices.

dsposv first attempts to factorize the matrix in SINGLE PRECISION and use this factorization within an iterative refinement procedure to produce a solution with DOUBLE PRECISION normwise backward error quality (see below). If the approach fails the method switches to a DOUBLE PRECISION factorization and solve.

The iterative refinement is not going to be a winning strategy if the ratio SINGLE PRECISION performance over DOUBLE PRECISION performance is too small. A reasonable strategy should take the number of right-hand sides and the size of the matrix into account. This might be done with a call to ILAENV in the future. Up to now, we always try iterative refinement.

The iterative refinement process is stopped if

```
ITER > ITERMAX
```

or for all the RHS we have:

```
RNRM < SQRT(N)*XNRM*ANRM*EPS*BWDMAX
```

where

```
o ITER is the number of the current iteration in the iterative
  refinement process
o RNRM is the infinity-norm of the residual
o XNRM is the infinity-norm of the solution
o ANRM is the infinity-operator-norm of the matrix A
o EPS is the machine epsilon returned by DLAMCH('Epsilon')
```

The value ITERMAX and BWDMAX are fixed to 30 and 1.0D+00 respectively.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsposv(UPLO, N, NRHS, A, LDA, B, LDB, X, LDX, WORK, SWORK, ITER,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dsposv_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             double *a, const armpl_int_t *lda, const double *b,
             const armpl_int_t *ldb, double *x, const armpl_int_t *ldx,
             double *work, float *swork, armpl_int_t *iter, armpl_int_t *info,
             ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array,. dimension (LDA, N) On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, if iterative refinement has been successfully used (INFO.EQ.0 and ITER.GE.0, see description below), then A is unchanged, if double precision factorization has been used (INFO.EQ.0 and ITER.LT.0, see description below), then the array A contains the factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T .

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). The N-by-NRHS right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, NRHS). If INFO = 0, the N-by-NRHS solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (N, NRHS). This array is used to hold the residual vectors.

SWORK Output parameter.

SWORK is REAL

SWORK is an array, dimension (N*(N+NRHS)). This array is used to use the single precision matrix and the right-hand sides or solutions in single precision.

ITER Output parameter.

ITER is INTEGER

< 0: iterative refinement has failed, double precision factorization has been performed -1 : the routine fell back to full precision for implementation- or machine-specific reasons -2 : narrowing the precision induced an overflow, the routine fell back to full precision -3 : failure of SPOTRF -31: stop the iterative refinement after the 30th iterations > 0: iterative refinement has been successfully used. Returns the number of iterations

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i of (DOUBLE PRECISION) A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

Related Information

It also exists with a native C interface as *LAPACKE_dsposv*.

4.6.82 dspsv

dspsv computes the solution to a real system of linear equations

```
A * X = B,
```

where A is an N-by-N symmetric matrix stored in packed format and X and B are N-by-NRHS matrices.

The diagonal pivoting method is used to factor A as

```
sA = U * D * U**T,  if UPLO = 'U', or
A = L * D * L**T,  if UPLO = 'L',
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine dspsv(UPLO, N, NRHS, AP, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void dspsv_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
            double *ap, armpl_int_t *ipiv, double *b, const armpl_int_t *ldb,
            armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AP Input and output parameter.

AP is DOUBLE PRECISION

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n. See below for further details.

On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by DSPTRF, stored as a packed triangular matrix in the same storage format as A.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D, as determined by DSPTRF. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged, and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, so the solution could not be computed.

Related Information

For this routine in other precisions, please see *cspsv*, *sspsv* and *zspsv*. It also exists with a native C interface as *LAPACKE_dspsv*.

4.6.83 dspsvx

dspsvx uses the diagonal pivoting factorization A = U*D*U^T or A = L*D*L^T to compute the solution to a real system of linear equations A * X = B, where A is an N-by-N symmetric matrix stored in packed format and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed:
```

1. If FACT = 'N', the diagonal pivoting method is used to factor A as

```
   A = U * D * U**T,  if UPLO = 'U', or
   A = L * D * L**T,  if UPLO = 'L',
where U (or L) is a product of permutation and unit upper (lower)
triangular matrices and D is symmetric and block diagonal with
1-by-1 and 2-by-2 diagonal blocks.
```

2. If some D(i,i)=0, so that D is exactly singular, then the routine

```
returns with INFO = i. Otherwise, the factored form of A is used
to estimate the condition number of the matrix A.  If the
reciprocal of the condition number is less than machine precision,
INFO = N+1 is returned as a warning, but the routine still goes on
to solve for X and compute error bounds as described below.
```

3. The system of equations is solved for X using the factored form

```
of A.
```

4. Iterative refinement is applied to improve the computed solution

```
 matrix and calculate error bounds and backward error estimates
for it.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dspsvx(FACT, UPLO, N, NRHS, AP, AFP, IPIV, B, LDB, X, LDX, RCOND,
                  FERR, BERR, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dspsvx_(const char *fact, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *nrhs, const double *ap, double *afp,
             armpl_int_t *ipiv, const double *b, const armpl_int_t *ldb,
             double *x, const armpl_int_t *ldx, double *rcond, double *ferr,
             double *berr, double *work, armpl_int_t *iwork,
             armpl_int_t *info, ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of A has been supplied on entry. = 'F': On entry, AFP and IPIV contain the factored form of A. AP, AFP and IPIV will not be modified. = 'N': The matrix A will be copied to AFP and factored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AP Input parameter.

AP is DOUBLE PRECISION

AP is an array, dimension (N*(N+1)/2). The upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n. See below for further details.

AFP Input and output parameter.

AFP is DOUBLE PRECISION

AFP is an array, dimension (N*(N+1)/2). If FACT = 'F', then AFP is an input argument and on entry contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by DSPTRF, stored as a packed triangular matrix in the same storage format as A.

If FACT = 'N', then AFP is an output argument and on exit contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by DSPTRF, stored as a packed triangular matrix in the same storage format as A.

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains details of the interchanges and the block structure of D, as determined by DSPTRF. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If

UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

If FACT = 'N', then IPIV is an output argument and on exit contains details of the interchanges and the block structure of D, as determined by DSPTRF.

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). The N-by-NRHS right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The estimate of the reciprocal condition number of the matrix A. If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: D(i,i) is exactly zero. The factorization has been completed but the factor D is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+1: D is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless,

the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *cspsvx*, *sspsvx* and *zspsvx*. It also exists with a native C interface as *LAPACKE_dspsvx*.

4.6.84 dsptrs

dsptrs solves a system of linear equations A*X = B with a real symmetric matrix A stored in packed format using the factorization A = U*D*U^T or A = L*D*L^T computed by DSPTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsptrs(UPLO, N, NRHS, AP, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void dsptrs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const double *ap, const armpl_int_t *ipiv, double *b,
             const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AP Input parameter.

AP is DOUBLE PRECISION

AP is an array, dimension (N*(N+1)/2). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by DSPTRF, stored as a packed triangular matrix.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by DSPTRF.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csptrs*, *ssptrs* and *zsptrs*. It also exists with a native C interface as *LAPACKE_dsptrs*.

4.6.85 dsysv

dsysv computes the solution to a real system of linear equations

```
A * X = B,
```

where A is an N-by-N symmetric matrix and X and B are N-by-NRHS matrices.

The diagonal pivoting method is used to factor A as

```
A = U * D * U**T,  if UPLO = 'U', or
A = L * D * L**T,  if UPLO = 'L',
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsysv(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dsysv_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
            double *a, const armpl_int_t *lda, armpl_int_t *ipiv, double *b,
            const armpl_int_t *ldb, double *work, const armpl_int_t *lwork,
            armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by DSYTRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D, as determined by DSYTRF. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged, and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= 1, and for best performance LWORK >= max(1,N*NB), where NB is the optimal blocksize for DSYTRF. for LWORK < N, TRS will be done with Level BLAS 2 for LWORK >= N, TRS will be done with Level BLAS 3

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, so the solution could not be computed.

Related Information

For this routine in other precisions, please see *csysv*, *ssysv* and *zsysv*. It also exists with a native C interface as *LAPACKE_dsysv*.

4.6.86 dsysv_aa

DSYSV computes the solution to a real system of linear equations

```
A * X = B,
```

where A is an N-by-N symmetric matrix and X and B are N-by-NRHS matrices.

Aasen's algorithm is used to factor A as

```
A = U * T * U**T,  if UPLO = 'U', or
A = L * T * L**T,  if UPLO = 'L',
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and T is symmetric tridiagonal. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsysv_aa(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dsysv_aa_(const char *uplo, const armpl_int_t *n,
               const armpl_int_t *nrhs, double *a, const armpl_int_t *lda,
               armpl_int_t *ipiv, double *b, const armpl_int_t *ldb,
               double *work, const armpl_int_t *lwork, armpl_int_t *info,
               ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the tridiagonal matrix T and the multipliers used to obtain the factor U or L from the factorization A = U*T*U^T or A = L*T*L^T as computed by DSYTRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

On exit, it contains the details of the interchanges, i.e., the row and column k of A were interchanged with the row and column IPIV(k).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= MAX(1,2*N,3*N-2), and for the best performance, LWORK >= MAX(1,N*NB), where NB is the optimal blocksize for DSYTRF_AA.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, so the solution could not be computed.

Related Information

For this routine in other precisions, please see *csysv_aa*, *ssysv_aa* and *zsysv_aa*. It also exists with a native C interface as *LAPACKE_dsysv_aa*.

4.6.87 dsysv_rk

DSYSV_RK computes the solution to a real system of linear equations A * X = B, where A is an N-by-N symmetric matrix and X and B are N-by-NRHS matrices.

The bounded Bunch-Kaufman (rook) diagonal pivoting method is used to factor A as

```
A = P*U*D*(U**T)*(P**T),  if UPLO = 'U', or
A = P*L*D*(L**T)*(P**T),  if UPLO = 'L',
```

where U (or L) is unit upper (or lower) triangular matrix, U^T (or L^T) is the transpose of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

DSYTRF_RK is called to compute the factorization of a real symmetric matrix. The factored form of A is then used to solve the system of equations A * X = B by calling BLAS3 routine DSYTRS_3.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsysv_rk(UPLO, N, NRHS, A, LDA, E, IPIV, B, LDB, WORK, LWORK,
                    INFO)
```

C specification:

```
#include "armpl.h"

void dsysv_rk_(const char *uplo, const armpl_int_t *n,
               const armpl_int_t *nrhs, double *a, const armpl_int_t *lda,
               double *e, armpl_int_t *ipiv, double *b,
               const armpl_int_t *ldb, double *work, const armpl_int_t *lwork,
               armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U': the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced.

If UPLO = 'L': the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, diagonal of the block diagonal matrix D and factors U or L as computed by DSYTRF_RK: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

For more info see the description of DSYTRF_RK routine.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N). On exit, contains the output computed by the factorization routine DSYTRF_RK, i.e. the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) is set to 0; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) is set to 0.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is set to 0 in both UPLO = 'U' or UPLO = 'L' cases.

For more info see the description of DSYTRF_RK routine.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D, as determined by DSYTRF_RK.

For more info see the description of DSYTRF_RK routine.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)).. Work array used in the factorization stage. On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= 1. For best performance of factorization stage LWORK >= max(1,N*NB), where NB is the optimal blocksize for DSYTRF_RK.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array for factorization stage, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit

< 0: If INFO = -k, the k-th argument had an illegal value

> 0: If INFO = k, the matrix A is singular, because: If UPLO = 'U': column k in the upper triangular part of A contains all zeros. If UPLO = 'L': column k in the lower triangular part of A contains all zeros.

Therefore D(k,k) is exactly zero, and superdiagonal elements of column k of U (or subdiagonal elements of column k of L) are all zeros. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

NOTE: INFO only stores the first occurrence of a singularity, any subsequent occurrence of singularity is not stored in INFO even though the factorization always completes.

Related Information

For this routine in other precisions, please see *csysv_rk*, *ssysv_rk* and *zsysv_rk*. It also exists with a native C interface as *LAPACKE_dsysv_rk*.

4.6.88 dsysv_rook

DSYSV_ROOK computes the solution to a real system of linear equations

```
A * X = B,
```

where A is an N-by-N symmetric matrix and X and B are N-by-NRHS matrices.

The diagonal pivoting method is used to factor A as

```
A = U * D * U**T,  if UPLO = 'U', or
A = L * D * L**T,  if UPLO = 'L',
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

DSYTRF_ROOK is called to compute the factorization of a real symmetric matrix A using the bounded Bunch-Kaufman ("rook") diagonal pivoting method.

The factored form of A is then used to solve the system of equations A * X = B by calling DSYTRS_ROOK.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsysv_rook(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dsysv_rook_(const char *uplo, const armpl_int_t *n,
                 const armpl_int_t *nrhs, double *a, const armpl_int_t *lda,
                 armpl_int_t *ipiv, double *b, const armpl_int_t *ldb,
                 double *work, const armpl_int_t *lwork, armpl_int_t *info,
                 ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by DSYTRF_ROOK.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D, as determined by DSYTRF_ROOK.

If UPLO = 'U': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k-1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k-1 and -IPIV(k-1) were inerchaged, D(k-1:k,k-1:k) is a 2-by-2 diagonal block.

If UPLO = 'L': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k+1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k+1 and -IPIV(k+1) were inerchaged, D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= 1, and for best performance LWORK >= max(1,N*NB), where NB is the optimal blocksize for DSYTRF_ROOK.

TRS will be done with Level 2 BLAS

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, so the solution could not be computed.

Related Information

For this routine in other precisions, please see *csysv_rook*, *ssysv_rook* and *zsysv_rook*. It also exists with a native C interface as *LAPACKE_dsysv_rook*.

4.6.89 dsysvx

dsysvx uses the diagonal pivoting factorization to compute the solution to a real system of linear equations A * X = B, where A is an N-by-N symmetric matrix and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed:
```

1. If FACT = 'N', the diagonal pivoting method is used to factor A.

```
 The form of the factorization is
   A = U * D * U**T,  if UPLO = 'U', or
   A = L * D * L**T,  if UPLO = 'L',
where U (or L) is a product of permutation and unit upper (lower)
triangular matrices, and D is symmetric and block diagonal with
1-by-1 and 2-by-2 diagonal blocks.
```

2. If some D(i,i)=0, so that D is exactly singular, then the routine

```
 returns with INFO = i. Otherwise, the factored form of A is used
to estimate the condition number of the matrix A.  If the
reciprocal of the condition number is less than machine precision,
INFO = N+1 is returned as a warning, but the routine still goes on
to solve for X and compute error bounds as described below.
```

3. The system of equations is solved for X using the factored form

```
of A.
```

4. Iterative refinement is applied to improve the computed solution

```
 matrix and calculate error bounds and backward error estimates
for it.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dsysvx(FACT, UPLO, N, NRHS, A, LDA, AF, LDAF, IPIV, B, LDB, X, LDX,
                  RCOND, FERR, BERR, WORK, LWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dsysvx_(const char *fact, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *nrhs, const double *a, const armpl_int_t *lda,
             double *af, const armpl_int_t *ldaf, armpl_int_t *ipiv,
             const double *b, const armpl_int_t *ldb, double *x,
             const armpl_int_t *ldx, double *rcond, double *ferr,
             double *berr, double *work, const armpl_int_t *lwork,
             armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of A has been supplied on entry. = 'F': On entry, AF and IPIV contain the factored form of A. AF and IPIV will not be modified. = 'N': The matrix A will be copied to AF and factored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). The symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input and output parameter.

AF is DOUBLE PRECISION

AF is an array, dimension (LDAF, N). If FACT = 'F', then AF is an input argument and on entry contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by DSYTRF.

If FACT = 'N', then AF is an output argument and on exit returns the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T .

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains details of the interchanges and the block structure of D, as determined by DSYTRF. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

If FACT = 'N', then IPIV is an output argument and on exit contains details of the interchanges and the block structure of D, as determined by DSYTRF.

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). The N-by-NRHS right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The estimate of the reciprocal condition number of the matrix A. If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= max(1,3*N), and for best performance, when FACT = 'N', LWORK >= max(1,3*N,N*NB), where NB is the optimal blocksize for DSYTRF.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: D(i,i) is exactly zero. The factorization has been completed but the factor D is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+1: D is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *csysvx*, *ssysvx* and *zsysvx*. It also exists with a native C interface as *LAPACKE_dsysvx*.

4.6.90 dsysvxx

```
DSYSVXX uses the diagonal pivoting factorization to compute the
   solution to a double precision system of linear equations A * X = B, where A
   is an N-by-N symmetric matrix and X and B are N-by-NRHS matrices.

   If requested, both normwise and maximum componentwise error bounds
   are returned. DSYSVXX will return a solution with a tiny
   guaranteed error (O(eps) where eps is the working machine
   precision) unless the matrix is very ill-conditioned, in which
   case a warning is returned. Relevant condition numbers also are
   calculated and returned.

   DSYSVXX accepts user-provided factorizations and equilibration
   factors; see the definitions of the FACT and EQUED options.
   Solving with refinement and using a factorization from a previous
   DSYSVXX call will also produce a solution with either O(eps)
   errors or warnings, but we cannot make that claim for general
   user-provided factorizations and equilibration factors if they
   differ from what DSYSVXX would itself produce.

                                                                    The␣
↪following steps are performed:

   1. If FACT = 'E', double precision scaling factors are computed to equilibrate
   the system:

     diag(S)*A*diag(S)      *inv(diag(S))*X = diag(S)*B

   Whether or not the system will be equilibrated depends on the
   scaling of the matrix A, but if equilibration is used, A is
   overwritten by diag(S)*A*diag(S) and B by diag(S)*B.

   2. If FACT = 'N' or 'E', the LU decomposition is used to factor
   the matrix A (after equilibration if FACT = 'E') as

      A = U * D * U**T,  if UPLO = 'U', or
      A = L * D * L**T,  if UPLO = 'L',

   where U (or L) is a product of permutation and unit upper (lower)
   triangular matrices, and D is symmetric and block diagonal with
   1-by-1 and 2-by-2 diagonal blocks.

   3. If some D(i,i)=0, so that D is exactly singular, then the
   routine returns with INFO = i. Otherwise, the factored form of A
   is used to estimate the condition number of the matrix A (see
   argument RCOND).  If the reciprocal of the condition number is
   less than machine precision, the routine still goes on to solve
   for X and compute error bounds as described below.

   4. The system of equations is solved for X using the factored form
   of A.

   5. By default (unless PARAMS(LA_LINRX_ITREF_I) is set to zero),
   the routine will use iterative refinement to try to get a small
   error and error bounds.  Refinement calculates the residual to at
   least twice the working precision.

   6. If equilibration was used, the matrix X is premultiplied by
   diag(R) so that it solves the original system before
   equilibration.                          Some optional parameters are bundled␣
↪in the PARAMS array.  These
```

(continues on next page)

(continued from previous page)

```
    settings determine how refinement is performed, but often the
    defaults are acceptable.  If the defaults are acceptable, users
    can pass NPARAMS = 0 which prevents the source code from accessing
    the PARAMS argument.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dsysvxx(FACT, UPLO, N, NRHS, A, LDA, AF, LDAF, IPIV, EQUED, S, B,
                   LDB, X, LDX, RCOND, RPVGRW, BERR, N_ERR_BNDS,
                   ERR_BNDS_NORM, ERR_BNDS_COMP, NPARAMS, PARAMS, WORK, IWORK,
                   INFO)
```

C specification:

```
#include "armpl.h"

void dsysvxx_(const char *fact, char *uplo, const armpl_int_t *n,
              const armpl_int_t *nrhs, double *a, const armpl_int_t *lda,
              double *af, const armpl_int_t *ldaf, armpl_int_t *ipiv,
              char *equed, double *s, double *b, const armpl_int_t *ldb,
              double *x, const armpl_int_t *ldx, double *rcond,
              double *rpvgrw, double *berr, const armpl_int_t *n_err_bnds,
              double *err_bnds_norm, double *err_bnds_comp,
              const armpl_int_t *nparams, double *params, double *work,
              armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AF and IPIV contain the factored form of A. If EQUED is not 'N', the matrix A has been equilibrated with scaling factors given by S. A, AF, and IPIV are not modified. = 'N': The matrix A will be copied to AF and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AF and factored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). The symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if FACT = 'E' and EQUED = 'Y', A is overwritten by diag(S)*A*diag(S).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input and output parameter.

AF is DOUBLE PRECISION

AF is an array, dimension (LDAF, N). If FACT = 'F', then AF is an input argument and on entry contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by DSYTRF.

If FACT = 'N', then AF is an output argument and on exit returns the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T .

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains details of the interchanges and the block structure of D, as determined by DSYTRF. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

If FACT = 'N', then IPIV is an output argument and on exit contains details of the interchanges and the block structure of D, as determined by DSYTRF.

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'Y': Both row and column equilibration, i.e., A has been replaced by diag(S) * A * diag(S). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

S Input and output parameter.

S is DOUBLE PRECISION

S is an array, dimension (N). The scale factors for A. If EQUED = 'Y', A is multiplied on the left and right by diag(S). S is an input argument if FACT = 'F'; otherwise, S is an output argument. If FACT = 'F' and EQUED = 'Y', each element of S must be positive. If S is output, each element of S is a power of the radix. If S is input, each element of S should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if EQUED = 'N', B is not modified; if EQUED = 'Y', B is overwritten by diag(S)*B;

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, NRHS). If INFO = 0, the N-by-NRHS solution matrix X to the original system of equations. Note that A and B are modified on exit if EQUED .ne. 'N', and the solution to the equilibrated system is inv(diag(S))*X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

RPVGRW Output parameter.

RPVGRW is DOUBLE PRECISION

Reciprocal pivot growth. On exit, this contains the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If this is much less than 1, then the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, estimated condition numbers, and error bounds could be unreliable. If factorization fails with 0<INFO<=N, then this contains the reciprocal pivot growth factor for the leading INFO columns of A.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). Componentwise relative backward error. This is the componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

N_ERR_BNDS Input parameter.

N_ERR_BNDS is INTEGER

Number of error bounds to return for each right hand side and each type (normwise or componentwise). See ERR_BNDS_NORM and ERR_BNDS_COMP below.

ERR_BNDS_NORM Output parameter.

ERR_BNDS_NORM is DOUBLE PRECISION

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 “Trust/don’t trust” boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * dlamch(‘Epsilon’).

err = 2 “Guaranteed” error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * dlamch(‘Epsilon’). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * dlamch(‘Epsilon’) to determine if the error estimate is “guaranteed”. These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Output parameter.

ERR_BNDS_COMP is DOUBLE PRECISION

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:, N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 “Trust/don’t trust” boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * dlamch(‘Epsilon’).

err = 2 “Guaranteed” error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * dlamch(‘Epsilon’). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * dlamch(‘Epsilon’) to determine if the error estimate is “guaranteed”. These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

NPARAMS Input parameter.

NPARAMS is INTEGER

Specifies the number of parameters set in PARAMS. If .LE. 0, the PARAMS array is never referenced and default values are used.

PARAMS Input and output parameter.

PARAMS is DOUBLE PRECISION

PARAMS is an array, dimension (NPARAMS). Specifies algorithm parameters. If an entry is .LT. 0.0, then that entry will be filled with default value used for that parameter. Only positions up to NPARAMS are accessed; defaults are used for higher-numbered parameters.

PARAMS(LA_LINRX_ITREF_I = 1) : Whether to perform iterative refinement or not. Default: 1.0D+0 = 0.0 : No refinement is performed, and no error bounds are computed. = 1.0 : Use the extra-precise refinement algorithm. (other values are reserved for future use)

PARAMS(LA_LINRX_ITHRESH_I = 2) : Maximum number of residual computations allowed for refinement. Default: 10 Aggressive: Set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in err_bnds_norm and err_bnds_comp may no longer be trustworthy.

PARAMS(LA_LINRX_CWISE_I = 3) : Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Default: 1.0 (attempt componentwise convergence)

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (4*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. The solution to every right-hand side is guaranteed. < 0: If INFO = -i, the i-th argument had an illegal value > 0 and <= N: U(INFO, INFO) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+J: The solution corresponding to the Jth right-hand side is not guaranteed. The solutions corresponding to other right- hand sides K with K > J may not be guaranteed as well, but only the first such right-hand side is reported. If a small componentwise error is not requested (PARAMS(3) = 0.0) then the Jth right-hand side is the first with a normwise error bound that is not guaranteed (the smallest J such that ERR_BNDS_NORM(J,1) = 0.0). By default (PARAMS(3) = 1.0) the Jth right-hand side is the first with either a normwise or componentwise error bound that is not guaranteed (the smallest J such that either ERR_BNDS_NORM(J,1) = 0.0 or ERR_BNDS_COMP(J,1) = 0.0). See the definition of ERR_BNDS_NORM(:,1) and ERR_BNDS_COMP(:,1). To get information about all of the right-hand sides check ERR_BNDS_NORM or ERR_BNDS_COMP.

Related Information

For this routine in other precisions, please see *csysvxx*, *ssysvxx* and *zsysvxx*. It also exists with a native C interface as *LAPACKE_dsysvxx*.

4.6.91 dsytrs

dsytrs solves a system of linear equations A*X = B with a real symmetric matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by DSYTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsytrs(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void dsytrs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const double *a, const armpl_int_t *lda, const armpl_int_t *ipiv,
             double *b, const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by DSYTRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by DSYTRF.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csytrs*, *ssytrs* and *zsytrs*. It also exists with a native C interface as *LAPACKE_dsytrs*.

4.6.92 dsytrs2

dsytrs2 solves a system of linear equations A*X = B with a real symmetric matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by DSYTRF and converted by DSYCONV.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsytrs2(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dsytrs2_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
              const double *a, const armpl_int_t *lda,
              const armpl_int_t *ipiv, double *b, const armpl_int_t *ldb,
              double *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by DSYTRF. Note that A is input / output. This might be counter-intuitive, and one may think that A is input only. A is input / output. This is because, at the start of the subroutine, we permute A in a "better" form and then we permute A back to its original form at the end.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by DSYTRF.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csytrs2*, *ssytrs2* and *zsytrs2*. It also exists with a native C interface as *LAPACKE_dsytrs2*.

4.6.93 dsytrs_3

DSYTRS_3 solves a system of linear equations A * X = B with a real symmetric matrix A using the factorization computed by DSYTRF_RK or DSYTRF_BK:

```
A = P*U*D*(U**T)*(P**T) or A = P*L*D*(L**T)*(P**T),
```

where U (or L) is unit upper (or lower) triangular matrix, U^T (or L^T) is the transpose of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This algorithm is using Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsytrs_3(UPLO, N, NRHS, A, LDA, E, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void dsytrs_3_(const char *uplo, const armpl_int_t *n,
               const armpl_int_t *nrhs, const double *a,
               const armpl_int_t *lda, const double *e,
               const armpl_int_t *ipiv, double *b, const armpl_int_t *ldb,
               armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix: = 'U': Upper triangular, form is A = P*U*D*(U^T)*(P^T); = 'L': Lower triangular, form is A = P*L*D*(L^T)*(P^T).

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). Diagonal of the block diagonal matrix D and factors U or L as computed by DSYTRF_RK and DSYTRF_BK: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D should be provided on entry in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Input parameter.

E is DOUBLE PRECISION

E is an array, dimension (N). On entry, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i),i=2:N, E(1) not referenced; If UPLO = 'L': E(i) = D(i+1,i),i=1:N-1, E(N) not referenced.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is not referenced in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by DSYTRF_RK or DSYTRF_BK.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csytrs_3*, *ssytrs_3* and *zsytrs_3*. It also exists with a native C interface as *LAPACKE_dsytrs_3*.

4.6.94 dsytrs_aa

DSYTRS_AA solves a system of linear equations A*X = B with a real symmetric matrix A using the factorization $A = U*T*U^T$ or $A = L*T*L^T$ computed by DSYTRF_AA.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsytrs_aa(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dsytrs_aa_(const char *uplo, const armpl_int_t *n,
                const armpl_int_t *nrhs, const double *a,
                const armpl_int_t *lda, const armpl_int_t *ipiv, double *b,
                const armpl_int_t *ldb, const double *work,
                const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*T*U^T ; = 'L': Lower triangular, form is A = L*T*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). Details of factors computed by DSYTRF_AA.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges as computed by DSYTRF_AA.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Input parameter.

WORK is DOUBLE array, dimension (MAX(1, LWORK))

LWORK Input parameter.

LWORK is INTEGER, LWORK >= MAX(1,3*N-2).

param[out] INFO verbatim INFO is INTEGER = 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csytrs_aa*, *ssytrs_aa* and *zsytrs_aa*. It also exists with a native C interface as *LAPACKE_dsytrs_aa*.

4.6.95 dsytrs_rook

DSYTRS_ROOK solves a system of linear equations A*X = B with a real symmetric matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by DSYTRF_ROOK.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsytrs_rook(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void dsytrs_rook_(const char *uplo, const armpl_int_t *n,
                  const armpl_int_t *nrhs, const double *a,
                  const armpl_int_t *lda, const armpl_int_t *ipiv, double *b,
                  const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by DSYTRF_ROOK.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by DSYTRF_ROOK.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csytrs_rook*, *ssytrs_rook* and *zsytrs_rook*. It also exists with a native C interface as *LAPACKE_dsytrs_rook*.

4.6.96 dtbtrs

dtbtrs solves a triangular system of the form

```
A * X = B  or  A**T * X = B,
```

where A is a triangular band matrix of order N, and B is an N-by NRHS matrix. A check is made to verify that A is nonsingular.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtbtrs(UPLO, TRANS, DIAG, N, KD, NRHS, AB, LDAB, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void dtbtrs_(const char *uplo, const char *trans, const char *diag,
             const armpl_int_t *n, const armpl_int_t *kd,
             const armpl_int_t *nrhs, const double *ab,
             const armpl_int_t *ldab, double *b, const armpl_int_t *ldb,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose = Transpose)

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals or subdiagonals of the triangular band matrix A. KD >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AB Input parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). The upper or lower triangular band matrix A, stored in the first kd+1 rows of AB. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd). If DIAG = 'U', the diagonal elements of A are not referenced and are assumed to be 1.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, if INFO = 0, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element of A is zero, indicating that the matrix is singular and the solutions X have not been computed.

Related Information

For this routine in other precisions, please see *ctbtrs*, *stbtrs* and *ztbtrs*. It also exists with a native C interface as *LAPACKE_dtbtrs*.

4.6.97 dtptrs

dtptrs solves a triangular system of the form

```
A * X = B  or  A**T * X = B,
```

where A is a triangular matrix of order N stored in packed format, and B is an N-by-NRHS matrix. A check is made to verify that A is nonsingular.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtptrs(UPLO, TRANS, DIAG, N, NRHS, AP, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void dtptrs_(const char *uplo, const char *trans, const char *diag,
             const armpl_int_t *n, const armpl_int_t *nrhs, const double *ap,
             double *b, const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose = Transpose)

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AP Input parameter.

AP is DOUBLE PRECISION

AP is an array, dimension (N*(N+1)/2). The upper or lower triangular matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, if INFO = 0, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element of A is zero, indicating that the matrix is singular and the solutions X have not been computed.

Related Information

For this routine in other precisions, please see *ctptrs*, *stptrs* and *ztptrs*. It also exists with a native C interface as *LAPACKE_dtptrs*.

4.6.98 sgbsv

sgbsv computes the solution to a real system of linear equations A * X = B, where A is a band matrix of order N with KL subdiagonals and KU superdiagonals, and X and B are N-by-NRHS matrices.

The LU decomposition with partial pivoting and row interchanges is used to factor A as A = L * U, where L is a product of permutation and unit lower triangular matrices with KL subdiagonals, and U is upper triangular with KL+KU superdiagonals. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgbsv(N, KL, KU, NRHS, AB, LDAB, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void sgbsv_(const armpl_int_t *n, const armpl_int_t *kl,
            const armpl_int_t *ku, const armpl_int_t *nrhs, float *ab,
            const armpl_int_t *ldab, armpl_int_t *ipiv, float *b,
            const armpl_int_t *ldb, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AB Input and output parameter.

AB is REAL

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows KL+1 to 2*KL+KU+1; rows 1 to KL of the array need not be set. The j-th column of A is stored in the j-th column of the array AB as follows: AB(KL+KU+1+i-j,j) = A(i,j) for max(1,j-KU)<=i<=min(N,j+KL) On exit, details of the factorization: U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1. See below for further details.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= 2*KL+KU+1.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices that define the permutation matrix P; row i of the matrix was interchanged with row IPIV(i).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and the solution has not been computed.

Related Information

For this routine in other precisions, please see *cgbsv*, *dgbsv* and *zgbsv*. It also exists with a native C interface as *LAPACKE_sgbsv*.

4.6.99 sgbsvx

sgbsvx uses the LU factorization to compute the solution to a real system of linear equations A * X = B, A^T * X = B, or A^H * X = B, where A is a band matrix of order N with KL subdiagonals and KU superdiagonals, and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed by this subroutine:
```

1. If FACT = 'E', real scaling factors are computed to equilibrate

```
 the system:
   TRANS = 'N':  diag(R)*A*diag(C)      *inv(diag(C))*X = diag(R)*B
   TRANS = 'T': (diag(R)*A*diag(C))**T *inv(diag(R))*X = diag(C)*B
   TRANS = 'C': (diag(R)*A*diag(C))**H *inv(diag(R))*X = diag(C)*B
Whether or not the system will be equilibrated depends on the
scaling of the matrix A, but if equilibration is used, A is
overwritten by diag(R)*A*diag(C) and B by diag(R)*B (if TRANS='N')
or diag(C)*B (if TRANS = 'T' or 'C').
```

2. If FACT = 'N' or 'E', the LU decomposition is used to factor the

```
 matrix A (after equilibration if FACT = 'E') as
   A = L * U,
where L is a product of permutation and unit lower triangular
matrices with KL subdiagonals, and U is upper triangular with
KL+KU superdiagonals.
```

3. If some U(i,i)=0, so that U is exactly singular, then the routine

```
 returns with INFO = i. Otherwise, the factored form of A is used
to estimate the condition number of the matrix A.  If the
reciprocal of the condition number is less than machine precision,
INFO = N+1 is returned as a warning, but the routine still goes on
to solve for X and compute error bounds as described below.
```

4. The system of equations is solved for X using the factored form

```
of A.
```

5. Iterative refinement is applied to improve the computed solution

```
 matrix and calculate error bounds and backward error estimates
for it.
```

6. If equilibration was used, the matrix X is premultiplied by

```
 diag(C) (if TRANS = 'N') or diag(R) (if TRANS = 'T' or 'C') so
that it solves the original system before equilibration.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine sgbsvx(FACT, TRANS, N, KL, KU, NRHS, AB, LDAB, AFB, LDAFB, IPIV,
                  EQUED, R, C, B, LDB, X, LDX, RCOND, FERR, BERR, WORK, IWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void sgbsvx_(const char *fact, const char *trans, const armpl_int_t *n,
             const armpl_int_t *kl, const armpl_int_t *ku,
             const armpl_int_t *nrhs, float *ab, const armpl_int_t *ldab,
             float *afb, const armpl_int_t *ldafb, armpl_int_t *ipiv,
             char *equed, float *r, float *c, float *b,
             const armpl_int_t *ldb, float *x, const armpl_int_t *ldx,
             float *rcond, float *ferr, float *berr, float *work,
             armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AFB and IPIV contain the factored form of A. If EQUED is not 'N', the matrix A has been equilibrated with scaling factors given by R and C. AB, AFB, and IPIV are not modified. = 'N': The matrix A will be copied to AFB and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AFB and factored.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations. = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Transpose)

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AB Input and output parameter.

AB is REAL

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(KU+1+i-j,j) = A(i,j) for max(1,j-KU)<=i<=min(N,j+kl)

If FACT = 'F' and EQUED is not 'N', then A must have been equilibrated by the scaling factors in R and/or C. AB is not modified if FACT = 'F' or 'N', or if FACT = 'E' and EQUED = 'N' on exit.

On exit, if EQUED .ne. 'N', A is scaled as follows: EQUED = 'R': A := diag(R) * A EQUED = 'C': A := A * diag(C) EQUED = 'B': A := diag(R) * A * diag(C).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KL+KU+1.

AFB Input and output parameter.

AFB is REAL

AFB is an array, dimension (LDAFB, N). If FACT = 'F', then AFB is an input argument and on entry contains details of the LU factorization of the band matrix A, as computed by SGBTRF. U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1. If EQUED .ne. 'N', then AFB is the factored form of the equilibrated matrix A.

If FACT = 'N', then AFB is an output argument and on exit returns details of the LU factorization of A.

If FACT = 'E', then AFB is an output argument and on exit returns details of the LU factorization of the equilibrated matrix A (see the description of AB for the form of the equilibrated matrix).

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AFB. LDAFB >= 2*KL+KU+1.

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains the pivot indices from the factorization A = L*U as computed by SGBTRF; row i of the matrix was interchanged with row IPIV(i).

If FACT = 'N', then IPIV is an output argument and on exit contains the pivot indices from the factorization A = L*U of the original matrix A.

If FACT = 'E', then IPIV is an output argument and on exit contains the pivot indices from the factorization A = L*U of the equilibrated matrix A.

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'R': Row equilibration, i.e., A has been premultiplied by diag(R). = 'C': Column equilibration, i.e., A has been postmultiplied by diag(C). = 'B': Both row and column equilibration, i.e., A has been replaced by diag(R) * A * diag(C). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

R Input and output parameter.

R is REAL

R is an array, dimension (N). The row scale factors for A. If EQUED = 'R' or 'B', A is multiplied on the left by diag(R); if EQUED = 'N' or 'C', R is not accessed. R is an input argument if FACT = 'F'; otherwise, R is an output argument. If FACT = 'F' and EQUED = 'R' or 'B', each element of R must be positive.

C Input and output parameter.

C is REAL

C is an array, dimension (N). The column scale factors for A. If EQUED = 'C' or 'B', A is multiplied on the right by diag(C); if EQUED = 'N' or 'R', C is not accessed. C is an input argument if FACT = 'F'; otherwise, C is an output argument. If FACT = 'F' and EQUED = 'C' or 'B', each element of C must be positive.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, if EQUED = 'N', B is not modified; if TRANS = 'N' and EQUED = 'R' or 'B', B is overwritten by diag(R)*B; if TRANS = 'T' or 'C' and EQUED = 'C' or 'B', B is overwritten by diag(C)*B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is REAL

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X to the original system of equations. Note that A and B are modified on exit if EQUED .ne. 'N', and the solution to the equilibrated system is inv(diag(C))* X if TRANS = 'N' and EQUED = 'C' or 'B', or inv(diag(R))* X if TRANS = 'T' or 'C' and EQUED = 'R' or 'B'.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

The estimate of the reciprocal condition number of the matrix A after equilibration (if done). If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (3*N). On exit, WORK(1) contains the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If WORK(1) is much less than 1, then

the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, condition estimator RCOND, and forward error bound FERR could be unreliable. If factorization fails with 0<INFO<=N, then WORK(1) contains the reciprocal pivot growth factor for the leading INFO columns of A.

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+1: U is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the

Related Information

For this routine in other precisions, please see *cgbsvx*, *dgbsvx* and *zgbsvx*. It also exists with a native C interface as *LAPACKE_sgbsvx*.

4.6.100 sgbsvxx

```
SGBSVXX uses the LU factorization to compute the solution to a
   real system of linear equations  A * X = B,  where A is an
   N-by-N matrix and X and B are N-by-NRHS matrices.

   If requested, both normwise and maximum componentwise error bounds
   are returned. SGBSVXX will return a solution with a tiny
   guaranteed error (O(eps) where eps is the working machine
   precision) unless the matrix is very ill-conditioned, in which
   case a warning is returned. Relevant condition numbers also are
   calculated and returned.

   SGBSVXX accepts user-provided factorizations and equilibration
   factors; see the definitions of the FACT and EQUED options.
   Solving with refinement and using a factorization from a previous
   SGBSVXX call will also produce a solution with either O(eps)
   errors or warnings, but we cannot make that claim for general
   user-provided factorizations and equilibration factors if they
   differ from what SGBSVXX would itself produce.

                                                                  The␣
↪following steps are performed:

   1. If FACT = 'E', real scaling factors are computed to equilibrate
   the system:

     TRANS = 'N':  diag(R)*A*diag(C)      *inv(diag(C))*X = diag(R)*B
     TRANS = 'T': (diag(R)*A*diag(C))**T  *inv(diag(R))*X = diag(C)*B
     TRANS = 'C': (diag(R)*A*diag(C))**H  *inv(diag(R))*X = diag(C)*B

   Whether or not the system will be equilibrated depends on the
   scaling of the matrix A, but if equilibration is used, A is
   overwritten by diag(R)*A*diag(C) and B by diag(R)*B (if TRANS='N')
   or diag(C)*B (if TRANS = 'T' or 'C').
```

(continues on next page)

(continued from previous page)

```
   2. If FACT = 'N' or 'E', the LU decomposition is used to factor
   the matrix A (after equilibration if FACT = 'E') as

     A = P * L * U,

   where P is a permutation matrix, L is a unit lower triangular
   matrix, and U is upper triangular.

   3. If some U(i,i)=0, so that U is exactly singular, then the
   routine returns with INFO = i. Otherwise, the factored form of A
   is used to estimate the condition number of the matrix A (see
   argument RCOND). If the reciprocal of the condition number is less
   than machine precision, the routine still goes on to solve for X
   and compute error bounds as described below.

   4. The system of equations is solved for X using the factored form
   of A.

   5. By default (unless PARAMS(LA_LINRX_ITREF_I) is set to zero),
   the routine will use iterative refinement to try to get a small
   error and error bounds.  Refinement calculates the residual to at
   least twice the working precision.

   6. If equilibration was used, the matrix X is premultiplied by
   diag(C) (if TRANS = 'N') or diag(R) (if TRANS = 'T' or 'C') so
   that it solves the original system before equilibration.
↪    Some optional parameters are bundled in the PARAMS array.  These
    settings determine how refinement is performed, but often the
    defaults are acceptable.  If the defaults are acceptable, users
    can pass NPARAMS = 0 which prevents the source code from accessing
    the PARAMS argument.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine sgbsvxx(FACT, TRANS, N, KL, KU, NRHS, AB, LDAB, AFB, LDAFB, IPIV,
                   EQUED, R, C, B, LDB, X, LDX, RCOND, RPVGRW, BERR,
                   N_ERR_BNDS, ERR_BNDS_NORM, ERR_BNDS_COMP, NPARAMS, PARAMS,
                   WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgbsvxx_(const char *fact, const char *trans, const armpl_int_t *n,
              const armpl_int_t *kl, const armpl_int_t *ku,
              const armpl_int_t *nrhs, float *ab, const armpl_int_t *ldab,
              float *afb, const armpl_int_t *ldafb, armpl_int_t *ipiv,
              char *equed, float *r, float *c, float *b,
              const armpl_int_t *ldb, float *x, const armpl_int_t *ldx,
              float *rcond, float *rpvgrw, float *berr,
              const armpl_int_t *n_err_bnds, float *err_bnds_norm,
              float *err_bnds_comp, const armpl_int_t *nparams, float *params,
              float *work, armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AF and IPIV contain the factored form of A. If EQUED is not 'N', the matrix A has been equilibrated with scaling factors given by R and C. A, AF, and IPIV are not modified. = 'N': The matrix A will be copied to AF and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AF and factored.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate Transpose = Transpose)

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AB Input and output parameter.

AB is REAL

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(KU+1+i-j,j) = A(i,j) for max(1,j-KU)<=i<=min(N,j+kl)

If FACT = 'F' and EQUED is not 'N', then AB must have been equilibrated by the scaling factors in R and/or C. AB is not modified if FACT = 'F' or 'N', or if FACT = 'E' and EQUED = 'N' on exit.

On exit, if EQUED .ne. 'N', A is scaled as follows: EQUED = 'R': A := diag(R) * A EQUED = 'C': A := A * diag(C) EQUED = 'B': A := diag(R) * A * diag(C).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KL+KU+1.

AFB Input and output parameter.

AFB is REAL

AFB is an array, dimension (LDAFB, N). If FACT = 'F', then AFB is an input argument and on entry contains details of the LU factorization of the band matrix A, as computed by SGBTRF. U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1. If EQUED .ne. 'N', then AFB is the factored form of the equilibrated matrix A.

If FACT = 'N', then AF is an output argument and on exit returns the factors L and U from the factorization A = P*L*U of the original matrix A.

If FACT = 'E', then AF is an output argument and on exit returns the factors L and U from the factorization A = P*L*U of the equilibrated matrix A (see the description of A for the form of the equilibrated matrix).

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AFB. LDAFB >= 2*KL+KU+1.

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains the pivot indices from the factorization A = P*L*U as computed by SGETRF; row i of the matrix was interchanged with row IPIV(i).

If FACT = 'N', then IPIV is an output argument and on exit contains the pivot indices from the factorization A = P*L*U of the original matrix A.

If FACT = 'E', then IPIV is an output argument and on exit contains the pivot indices from the factorization A = P*L*U of the equilibrated matrix A.

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'R': Row equilibration, i.e., A has been premultiplied by diag(R). = 'C': Column equilibration, i.e., A has been postmultiplied by diag(C). = 'B': Both row and column equilibration, i.e., A has been replaced by diag(R) * A * diag(C). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

R Input and output parameter.

R is REAL

R is an array, dimension (N). The row scale factors for A. If EQUED = 'R' or 'B', A is multiplied on the left by diag(R); if EQUED = 'N' or 'C', R is not accessed. R is an input argument if FACT = 'F'; otherwise, R is an output argument. If FACT = 'F' and EQUED = 'R' or 'B', each element of R must be positive. If R is output, each element of R is a power of the radix. If R is input, each element of R should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

C Input and output parameter.

C is REAL

C is an array, dimension (N). The column scale factors for A. If EQUED = 'C' or 'B', A is multiplied on the right by diag(C); if EQUED = 'N' or 'R', C is not accessed. C is an input argument if FACT = 'F'; otherwise, C is an output argument. If FACT = 'F' and EQUED = 'C' or 'B', each element of C must be positive. If C is output, each element of C is a power of the radix. If C is input, each element of C should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if EQUED = 'N', B is not modified; if TRANS = 'N' and EQUED = 'R' or 'B', B is overwritten by diag(R)*B; if TRANS = 'T' or 'C' and EQUED = 'C' or 'B', B is overwritten by diag(C)*B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is REAL

X is an array, dimension (LDX, NRHS). If INFO = 0, the N-by-NRHS solution matrix X to the original system of equations. Note that A and B are modified on exit if EQUED .ne. 'N', and the solution to the equilibrated system is inv(diag(C))* X if TRANS = 'N' and EQUED = 'C' or 'B', or inv(diag(R))* X if TRANS = 'T' or 'C' and EQUED = 'R' or 'B'.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

RPVGRW Output parameter.

RPVGRW is REAL

Reciprocal pivot growth. On exit, this contains the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If this is much less than 1, then the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, estimated condition numbers, and error bounds could be unreliable. If factorization fails with 0<INFO<=N, then this contains the reciprocal pivot growth factor for the leading INFO columns of A. In SGESVX, this quantity is returned in WORK(1).

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). Componentwise relative backward error. This is the componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

N_ERR_BNDS Input parameter.

N_ERR_BNDS is INTEGER

Number of error bounds to return for each right hand side and each type (normwise or componentwise). See ERR_BNDS_NORM and ERR_BNDS_COMP below.

ERR_BNDS_NORM Output parameter.

ERR_BNDS_NORM is REAL

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 “Guaranteed” error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch(‘Epsilon’). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch(‘Epsilon’) to determine if the error estimate is “guaranteed”. These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Output parameter.

ERR_BNDS_COMP is REAL

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:, N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 “Trust/don’t trust” boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch(‘Epsilon’).

err = 2 “Guaranteed” error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch(‘Epsilon’). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch(‘Epsilon’) to determine if the error estimate is “guaranteed”. These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

NPARAMS Input parameter.

NPARAMS is INTEGER

Specifies the number of parameters set in PARAMS. If .LE. 0, the PARAMS array is never referenced and default values are used.

PARAMS Input and output parameter.

PARAMS is REAL

PARAMS is an array, dimension NPARAMS. Specifies algorithm parameters. If an entry is .LT. 0.0, then that entry will be filled with default value used for that parameter. Only positions up to NPARAMS are accessed; defaults are used for higher-numbered parameters.

PARAMS(LA_LINRX_ITREF_I = 1) : Whether to perform iterative refinement or not. Default: 1.0 = 0.0 : No refinement is performed, and no error bounds are computed. = 1.0 : Use the double-precision refinement algorithm, possibly with doubled-single computations if the compilation environment does not support DOUBLE PRECISION. (other values are reserved for future use)

PARAMS(LA_LINRX_ITHRESH_I = 2) : Maximum number of residual computations allowed for refinement. Default: 10 Aggressive: Set to 100 to permit convergence using approximate factorizations or

factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in err_bnds_norm and err_bnds_comp may no longer be trustworthy.

PARAMS(LA_LINRX_CWISE_I = 3) : Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Default: 1.0 (attempt componentwise convergence)

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (4*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. The solution to every right-hand side is guaranteed. < 0: If INFO = -i, the i-th argument had an illegal value > 0 and <= N: U(INFO, INFO) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+J: The solution corresponding to the Jth right-hand side is not guaranteed. The solutions corresponding to other right- hand sides K with K > J may not be guaranteed as well, but only the first such right-hand side is reported. If a small componentwise error is not requested (PARAMS(3) = 0.0) then the Jth right-hand side is the first with a normwise error bound that is not guaranteed (the smallest J such that ERR_BNDS_NORM(J,1) = 0.0). By default (PARAMS(3) = 1.0) the Jth right-hand side is the first with either a normwise or componentwise error bound that is not guaranteed (the smallest J such that either ERR_BNDS_NORM(J,1) = 0.0 or ERR_BNDS_COMP(J,1) = 0.0). See the definition of ERR_BNDS_NORM(:,1) and ERR_BNDS_COMP(:,1). To get information about all of the right-hand sides check ERR_BNDS_NORM or ERR_BNDS_COMP.

Related Information

For this routine in other precisions, please see *cgbsvxx*, *dgbsvxx* and *zgbsvxx*. It also exists with a native C interface as *LAPACKE_sgbsvxx*.

4.6.101 sgbtrs

sgbtrs solves a system of linear equations

```
A * X = B  or  A**T * X = B
```

with a general band matrix A using the LU factorization computed by SGBTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgbtrs(TRANS, N, KL, KU, NRHS, AB, LDAB, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void sgbtrs_(const char *trans, const armpl_int_t *n, const armpl_int_t *kl,
             const armpl_int_t *ku, const armpl_int_t *nrhs, const float *ab,
```

(continues on next page)

(continued from previous page)

```
                const armpl_int_t *ldab, const armpl_int_t *ipiv, float *b,
                const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations. = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^T * X = B (Conjugate transpose = Transpose)

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AB Input parameter.

AB is REAL

AB is an array, dimension (LDAB, N). Details of the LU factorization of the band matrix A, as computed by SGBTRF. U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= 2*KL+KU+1.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices; for 1 <= i <= N, row i of the matrix was interchanged with row IPIV(i).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgbtrs*, *dgbtrs* and *zgbtrs*. It also exists with a native C interface as *LAPACKE_sgbtrs*.

4.6.102 sgesv

sgesv computes the solution to a real system of linear equations

```
A * X = B,
```

where A is an N-by-N matrix and X and B are N-by-NRHS matrices.

The LU decomposition with partial pivoting and row interchanges is used to factor A as

```
A = P * L * U,
```

where P is a permutation matrix, L is unit lower triangular, and U is upper triangular. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgesv(N, NRHS, A, LDA, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void sgesv_(const armpl_int_t *n, const armpl_int_t *nrhs, float *a,
            const armpl_int_t *lda, armpl_int_t *ipiv, float *b,
            const armpl_int_t *ldb, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the N-by-N coefficient matrix A. On exit, the factors L and U from the factorization A = P*L*U; the unit diagonal elements of L are not stored.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices that define the permutation matrix P; row i of the matrix was interchanged with row IPIV(i).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

Related Information

For this routine in other precisions, please see *cgesv*, *dgesv* and *zgesv*. It also exists with a native C interface as *LAPACKE_sgesv*.

4.6.103 sgesvx

sgesvx uses the LU factorization to compute the solution to a real system of linear equations

```
A * X = B,
```

where A is an N-by-N matrix and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed:
```

1. If FACT = 'E', real scaling factors are computed to equilibrate

```
 the system:
   TRANS = 'N':  diag(R)*A*diag(C)      *inv(diag(C))*X = diag(R)*B
   TRANS = 'T': (diag(R)*A*diag(C))**T  *inv(diag(R))*X = diag(C)*B
   TRANS = 'C': (diag(R)*A*diag(C))**H  *inv(diag(R))*X = diag(C)*B
Whether or not the system will be equilibrated depends on the
scaling of the matrix A, but if equilibration is used, A is
overwritten by diag(R)*A*diag(C) and B by diag(R)*B (if TRANS='N')
or diag(C)*B (if TRANS = 'T' or 'C').
```

2. If FACT = 'N' or 'E', the LU decomposition is used to factor the

```
 matrix A (after equilibration if FACT = 'E') as
   A = P * L * U,
where P is a permutation matrix, L is a unit lower triangular
matrix, and U is upper triangular.
```

3. If some U(i,i)=0, so that U is exactly singular, then the routine

```
 returns with INFO = i. Otherwise, the factored form of A is used
to estimate the condition number of the matrix A.  If the
reciprocal of the condition number is less than machine precision,
INFO = N+1 is returned as a warning, but the routine still goes on
to solve for X and compute error bounds as described below.
```

4. The system of equations is solved for X using the factored form

```
of A.
```

5. Iterative refinement is applied to improve the computed solution

```
 matrix and calculate error bounds and backward error estimates
for it.
```

6. If equilibration was used, the matrix X is premultiplied by

```
 diag(C) (if TRANS = 'N') or diag(R) (if TRANS = 'T' or 'C') so
that it solves the original system before equilibration.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine sgesvx(FACT, TRANS, N, NRHS, A, LDA, AF, LDAF, IPIV, EQUED, R, C,
                  B, LDB, X, LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgesvx_(const char *fact, const char *trans, const armpl_int_t *n,
             const armpl_int_t *nrhs, float *a, const armpl_int_t *lda,
             float *af, const armpl_int_t *ldaf, armpl_int_t *ipiv,
             char *equed, float *r, float *c, float *b,
             const armpl_int_t *ldb, float *x, const armpl_int_t *ldx,
             float *rcond, float *ferr, float *berr, float *work,
             armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AF and IPIV contain the factored form of A. If EQUED is not 'N', the matrix A has been equilibrated with scaling factors given by R and C. A, AF, and IPIV are not modified. = 'N': The matrix A will be copied to AF and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AF and factored.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Transpose)

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A. If FACT = 'F' and EQUED is not 'N', then A must have been equilibrated by the scaling factors in R and/or C. A is not modified if FACT = 'F' or 'N', or if FACT = 'E' and EQUED = 'N' on exit.

On exit, if EQUED .ne. 'N', A is scaled as follows: EQUED = 'R': A := diag(R) * A EQUED = 'C': A := A * diag(C) EQUED = 'B': A := diag(R) * A * diag(C).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input and output parameter.

AF is REAL

AF is an array, dimension (LDAF, N). If FACT = 'F', then AF is an input argument and on entry contains the factors L and U from the factorization A = P*L*U as computed by SGETRF. If EQUED .ne. 'N', then AF is the factored form of the equilibrated matrix A.

If FACT = 'N', then AF is an output argument and on exit returns the factors L and U from the factorization A = P*L*U of the original matrix A.

If FACT = 'E', then AF is an output argument and on exit returns the factors L and U from the factorization A = P*L*U of the equilibrated matrix A (see the description of A for the form of the equilibrated matrix).

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains the pivot indices from the factorization A = P*L*U as computed by SGETRF; row i of the matrix was interchanged with row IPIV(i).

If FACT = 'N', then IPIV is an output argument and on exit contains the pivot indices from the factorization A = P*L*U of the original matrix A.

If FACT = 'E', then IPIV is an output argument and on exit contains the pivot indices from the factorization A = P*L*U of the equilibrated matrix A.

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'R': Row equilibration, i.e., A has been premultiplied by diag(R). = 'C': Column equilibration, i.e., A has been postmultiplied by diag(C). = 'B': Both row and column equilibration, i.e., A has been replaced by diag(R) * A * diag(C). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

R Input and output parameter.

R is REAL

R is an array, dimension (N). The row scale factors for A. If EQUED = 'R' or 'B', A is multiplied on the left by diag(R); if EQUED = 'N' or 'C', R is not accessed. R is an input argument if FACT = 'F'; otherwise, R is an output argument. If FACT = 'F' and EQUED = 'R' or 'B', each element of R must be positive.

C Input and output parameter.

C is REAL

C is an array, dimension (N). The column scale factors for A. If EQUED = 'C' or 'B', A is multiplied on the right by diag(C); if EQUED = 'N' or 'R', C is not accessed. C is an input argument if FACT = 'F'; otherwise, C is an output argument. If FACT = 'F' and EQUED = 'C' or 'B', each element of C must be positive.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if EQUED = 'N', B is not modified; if TRANS = 'N' and EQUED = 'R' or 'B', B is overwritten by diag(R)*B; if TRANS = 'T' or 'C' and EQUED = 'C' or 'B', B is overwritten by diag(C)*B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is REAL

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X to the original system of equations. Note that A and B are modified on exit if EQUED .ne. 'N', and the solution to the equilibrated system is inv(diag(C))* X if TRANS = 'N' and EQUED = 'C' or 'B', or inv(diag(R))* X if TRANS = 'T' or 'C' and EQUED = 'R' or 'B'.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

The estimate of the reciprocal condition number of the matrix A after equilibration (if done). If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (4*N). On exit, WORK(1) contains the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If WORK(1) is much less than 1, then the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, condition estimator RCOND, and forward error bound FERR could be unreliable. If factorization fails with 0<INFO<=N, then WORK(1) contains the reciprocal pivot growth factor for the leading INFO columns of A.

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+1: U is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *cgesvx*, *dgesvx* and *zgesvx*. It also exists with a native C interface as *LAPACKE_sgesvx*.

4.6.104 sgesvxx

```
SGESVXX uses the LU factorization to compute the solution to a
   real system of linear equations  A * X = B,  where A is an
   N-by-N matrix and X and B are N-by-NRHS matrices.

   If requested, both normwise and maximum componentwise error bounds
   are returned. SGESVXX will return a solution with a tiny
   guaranteed error (O(eps) where eps is the working machine
   precision) unless the matrix is very ill-conditioned, in which
   case a warning is returned. Relevant condition numbers also are
   calculated and returned.

   SGESVXX accepts user-provided factorizations and equilibration
   factors; see the definitions of the FACT and EQUED options.
   Solving with refinement and using a factorization from a previous
   SGESVXX call will also produce a solution with either O(eps)
   errors or warnings, but we cannot make that claim for general
   user-provided factorizations and equilibration factors if they
   differ from what SGESVXX would itself produce.

                                                                  The␣
↪following steps are performed:

   1. If FACT = 'E', real scaling factors are computed to equilibrate
   the system:

     TRANS = 'N':  diag(R)*A*diag(C)     *inv(diag(C))*X = diag(R)*B
```

(continues on next page)

(continued from previous page)

```
     TRANS = 'T': (diag(R)*A*diag(C))**T *inv(diag(R))*X = diag(C)*B
     TRANS = 'C': (diag(R)*A*diag(C))**H *inv(diag(R))*X = diag(C)*B

   Whether or not the system will be equilibrated depends on the
   scaling of the matrix A, but if equilibration is used, A is
   overwritten by diag(R)*A*diag(C) and B by diag(R)*B (if TRANS='N')
   or diag(C)*B (if TRANS = 'T' or 'C').

   2. If FACT = 'N' or 'E', the LU decomposition is used to factor
   the matrix A (after equilibration if FACT = 'E') as

     A = P * L * U,

   where P is a permutation matrix, L is a unit lower triangular
   matrix, and U is upper triangular.

   3. If some U(i,i)=0, so that U is exactly singular, then the
   routine returns with INFO = i. Otherwise, the factored form of A
   is used to estimate the condition number of the matrix A (see
   argument RCOND). If the reciprocal of the condition number is less
   than machine precision, the routine still goes on to solve for X
   and compute error bounds as described below.

   4. The system of equations is solved for X using the factored form
   of A.

   5. By default (unless PARAMS(LA_LINRX_ITREF_I) is set to zero),
   the routine will use iterative refinement to try to get a small
   error and error bounds.  Refinement calculates the residual to at
   least twice the working precision.

   6. If equilibration was used, the matrix X is premultiplied by
   diag(C) (if TRANS = 'N') or diag(R) (if TRANS = 'T' or 'C') so
   that it solves the original system before equilibration.
↪   Some optional parameters are bundled in the PARAMS array.  These
    settings determine how refinement is performed, but often the
    defaults are acceptable.  If the defaults are acceptable, users
    can pass NPARAMS = 0 which prevents the source code from accessing
    the PARAMS argument.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine sgesvxx(FACT, TRANS, N, NRHS, A, LDA, AF, LDAF, IPIV, EQUED, R, C,
                   B, LDB, X, LDX, RCOND, RPVGRW, BERR, N_ERR_BNDS,
                   ERR_BNDS_NORM, ERR_BNDS_COMP, NPARAMS, PARAMS, WORK, IWORK,
                   INFO)
```

C specification:

```
#include "armpl.h"

void sgesvxx_(const char *fact, const char *trans, const armpl_int_t *n,
              const armpl_int_t *nrhs, float *a, const armpl_int_t *lda,
              float *af, const armpl_int_t *ldaf, armpl_int_t *ipiv,
              char *equed, float *r, float *c, float *b,
              const armpl_int_t *ldb, float *x, const armpl_int_t *ldx,
```

(continues on next page)

(continued from previous page)

```
                float *rcond, float *rpvgrw, float *berr,
                const armpl_int_t *n_err_bnds, float *err_bnds_norm,
                float *err_bnds_comp, const armpl_int_t *nparams, float *params,
                float *work, armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AF and IPIV contain the factored form of A. If EQUED is not 'N', the matrix A has been equilibrated with scaling factors given by R and C. A, AF, and IPIV are not modified. = 'N': The matrix A will be copied to AF and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AF and factored.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate Transpose = Transpose)

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A. If FACT = 'F' and EQUED is not 'N', then A must have been equilibrated by the scaling factors in R and/or C. A is not modified if FACT = 'F' or 'N', or if FACT = 'E' and EQUED = 'N' on exit.

On exit, if EQUED .ne. 'N', A is scaled as follows: EQUED = 'R': A := diag(R) * A EQUED = 'C': A := A * diag(C) EQUED = 'B': A := diag(R) * A * diag(C).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input and output parameter.

AF is REAL

AF is an array, dimension (LDAF, N). If FACT = 'F', then AF is an input argument and on entry contains the factors L and U from the factorization A = P*L*U as computed by SGETRF. If EQUED .ne. 'N', then AF is the factored form of the equilibrated matrix A.

If FACT = 'N', then AF is an output argument and on exit returns the factors L and U from the factorization A = P*L*U of the original matrix A.

If FACT = 'E', then AF is an output argument and on exit returns the factors L and U from the factorization A = P*L*U of the equilibrated matrix A (see the description of A for the form of the equilibrated matrix).

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains the pivot indices from the factorization A = P*L*U as computed by SGETRF; row i of the matrix was interchanged with row IPIV(i).

If FACT = 'N', then IPIV is an output argument and on exit contains the pivot indices from the factorization A = P*L*U of the original matrix A.

If FACT = 'E', then IPIV is an output argument and on exit contains the pivot indices from the factorization A = P*L*U of the equilibrated matrix A.

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'R': Row equilibration, i.e., A has been premultiplied by diag(R). = 'C': Column equilibration, i.e., A has been postmultiplied by diag(C). = 'B': Both row and column equilibration, i.e., A has been replaced by diag(R) * A * diag(C). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

R Input and output parameter.

R is REAL

R is an array, dimension (N). The row scale factors for A. If EQUED = 'R' or 'B', A is multiplied on the left by diag(R); if EQUED = 'N' or 'C', R is not accessed. R is an input argument if FACT = 'F'; otherwise, R is an output argument. If FACT = 'F' and EQUED = 'R' or 'B', each element of R must be positive. If R is output, each element of R is a power of the radix. If R is input, each element of R should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

C Input and output parameter.

C is REAL

C is an array, dimension (N). The column scale factors for A. If EQUED = 'C' or 'B', A is multiplied on the right by diag(C); if EQUED = 'N' or 'R', C is not accessed. C is an input argument if FACT = 'F'; otherwise, C is an output argument. If FACT = 'F' and EQUED = 'C' or 'B', each element of C must be positive. If C is output, each element of C is a power of the radix. If C is input, each element of C should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if EQUED = 'N', B is not modified; if TRANS = 'N' and EQUED = 'R' or 'B', B is overwritten by diag(R)*B; if TRANS = 'T' or 'C' and EQUED = 'C' or 'B', B is overwritten by diag(C)*B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is REAL

X is an array, dimension (LDX, NRHS). If INFO = 0, the N-by-NRHS solution matrix X to the original system of equations. Note that A and B are modified on exit if EQUED .ne. 'N', and the solution to the equilibrated system is inv(diag(C))* X if TRANS = 'N' and EQUED = 'C' or 'B', or inv(diag(R))* X if TRANS = 'T' or 'C' and EQUED = 'R' or 'B'.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

RPVGRW Output parameter.

RPVGRW is REAL

Reciprocal pivot growth. On exit, this contains the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If this is much less than 1, then the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, estimated condition numbers, and error bounds could be unreliable. If factorization fails with 0<INFO<=N, then this contains the reciprocal pivot growth factor for the leading INFO columns of A. In SGESVX, this quantity is returned in WORK(1).

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). Componentwise relative backward error. This is the componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

N_ERR_BNDS Input parameter.

N_ERR_BNDS is INTEGER

Number of error bounds to return for each right hand side and each type (normwise or componentwise). See ERR_BNDS_NORM and ERR_BNDS_COMP below.

ERR_BNDS_NORM Output parameter.

ERR_BNDS_NORM is REAL

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Output parameter.

ERR_BNDS_COMP is REAL

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:, N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

NPARAMS Input parameter.

NPARAMS is INTEGER

Specifies the number of parameters set in PARAMS. If .LE. 0, the PARAMS array is never referenced and default values are used.

PARAMS Input and output parameter.

PARAMS is REAL

PARAMS is an array, dimension NPARAMS. Specifies algorithm parameters. If an entry is .LT. 0.0, then that entry will be filled with default value used for that parameter. Only positions up to NPARAMS are accessed; defaults are used for higher-numbered parameters.

PARAMS(LA_LINRX_ITREF_I = 1) : Whether to perform iterative refinement or not. Default: 1.0 = 0.0 : No refinement is performed, and no error bounds are computed. = 1.0 : Use the double-precision refinement algorithm, possibly with doubled-single computations if the compilation environment does not support DOUBLE PRECISION. (other values are reserved for future use)

PARAMS(LA_LINRX_ITHRESH_I = 2) : Maximum number of residual computations allowed for refinement. Default: 10 Aggressive: Set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in err_bnds_norm and err_bnds_comp may no longer be trustworthy.

PARAMS(LA_LINRX_CWISE_I = 3) : Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Default: 1.0 (attempt componentwise convergence)

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (4*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. The solution to every right-hand side is guaranteed. < 0: If INFO = -i, the i-th argument had an illegal value > 0 and <= N: U(INFO, INFO) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+J: The solution corresponding to the Jth right-hand side is not guaranteed. The solutions corresponding to other right- hand sides K with K > J may not be guaranteed as well, but only the first such right-hand side is reported. If a small componentwise error is not requested (PARAMS(3) = 0.0) then the Jth right-hand side is the first with a normwise error bound that is not guaranteed (the smallest J such that ERR_BNDS_NORM(J,1) = 0.0). By default (PARAMS(3) = 1.0) the Jth right-hand side is the first with either a normwise or componentwise error bound that is not guaranteed (the smallest J such that either ERR_BNDS_NORM(J,1) = 0.0 or ERR_BNDS_COMP(J,1) = 0.0). See the definition of ERR_BNDS_NORM(:,1) and ERR_BNDS_COMP(:,1). To get information about all of the right-hand sides check ERR_BNDS_NORM or ERR_BNDS_COMP.

Related Information

For this routine in other precisions, please see *cgesvxx*, *dgesvxx* and *zgesvxx*. It also exists with a native C interface as *LAPACKE_sgesvxx*.

4.6.105 sgetrs

sgetrs solves a system of linear equations

```
A * X = B  or  A**T * X = B
```

with a general N-by-N matrix A using the LU factorization computed by SGETRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgetrs(TRANS, N, NRHS, A, LDA, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void sgetrs_(const char *trans, const armpl_int_t *n, const armpl_int_t *nrhs,
             const float *a, const armpl_int_t *lda, const armpl_int_t *ipiv,
             float *b, const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^T * X = B (Conjugate transpose = Transpose)

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). The factors L and U from the factorization A = P*L*U as computed by SGETRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from SGETRF; for 1<=i<=N, row i of the matrix was interchanged with row IPIV(i).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgetrs*, *dgetrs* and *zgetrs*. It also exists with a native C interface as *LAPACKE_sgetrs*.

4.6.106 sgtsv

sgtsv solves the equation

```
A*X = B,
```

where A is an n by n tridiagonal matrix, by Gaussian elimination with partial pivoting.

Note that the equation A^T *X = B may be solved by interchanging the order of the arguments DU and DL.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgtsv(N, NRHS, DL, D, DU, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void sgtsv_(const armpl_int_t *n, const armpl_int_t *nrhs, float *dl,
            float *d, float *du, float *b, const armpl_int_t *ldb,
            armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

DL Input and output parameter.

DL is REAL

DL is an array, dimension (N-1). On entry, DL must contain the (n-1) sub-diagonal elements of A.

On exit, DL is overwritten by the (n-2) elements of the second super-diagonal of the upper triangular matrix U from the LU factorization of A, in DL(1), . . . , DL(n-2).

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry, D must contain the diagonal elements of A.

On exit, D is overwritten by the n diagonal elements of U.

DU Input and output parameter.

DU is REAL

DU is an array, dimension (N-1). On entry, DU must contain the (n-1) super-diagonal elements of A.

On exit, DU is overwritten by the (n-1) elements of the first super-diagonal of U.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the N by NRHS matrix of right hand side matrix B. On exit, if INFO = 0, the N by NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, U(i,i) is exactly zero, and the solution has not been computed. The factorization has not been completed unless i = N.

Related Information

For this routine in other precisions, please see *cgtsv*, *dgtsv* and *zgtsv*. It also exists with a native C interface as *LAPACKE_sgtsv*.

4.6.107 sgtsvx

sgtsvx uses the LU factorization to compute the solution to a real system of linear equations A * X = B or A^T * X = B, where A is a tridiagonal matrix of order N and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed:
```

1. If FACT = 'N', the LU decomposition is used to factor the matrix A

```
 as A = L * U, where L is a product of permutation and unit lower
bidiagonal matrices and U is upper triangular with nonzeros in
only the main diagonal and first two superdiagonals.
```

2. If some U(i,i)=0, so that U is exactly singular, then the routine

```
 returns with INFO = i. Otherwise, the factored form of A is used
to estimate the condition number of the matrix A.  If the
reciprocal of the condition number is less than machine precision,
INFO = N+1 is returned as a warning, but the routine still goes on
to solve for X and compute error bounds as described below.
```

3. The system of equations is solved for X using the factored form

```
of A.
```

4. Iterative refinement is applied to improve the computed solution

```
 matrix and calculate error bounds and backward error estimates
for it.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine sgtsvx(FACT, TRANS, N, NRHS, DL, D, DU, DLF, DF, DUF, DU2, IPIV, B,
                  LDB, X, LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgtsvx_(const char *fact, const char *trans, const armpl_int_t *n,
             const armpl_int_t *nrhs, const float *dl, const float *d,
             const float *du, float *dlf, float *df, float *duf, float *du2,
             armpl_int_t *ipiv, const float *b, const armpl_int_t *ldb,
             float *x, const armpl_int_t *ldx, float *rcond, float *ferr,
             float *berr, float *work, armpl_int_t *iwork, armpl_int_t *info,
             ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of A has been supplied on entry. = 'F': DLF, DF, DUF, DU2, and IPIV contain the factored form of A; DL, D, DU, DLF, DF, DUF, DU2 and IPIV will not be modified. = 'N': The matrix will be copied to DLF, DF, and DUF and factored.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose = Transpose)

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

DL Input parameter.

DL is REAL

DL is an array, dimension (N-1). The (n-1) subdiagonal elements of A.

D Input parameter.

D is REAL

D is an array, dimension (N). The n diagonal elements of A.

DU Input parameter.

DU is REAL

DU is an array, dimension (N-1). The (n-1) superdiagonal elements of A.

DLF Input and output parameter.

DLF is REAL

DLF is an array, dimension (N-1). If FACT = 'F', then DLF is an input argument and on entry contains the (n-1) multipliers that define the matrix L from the LU factorization of A as computed by SGTTRF.

If FACT = 'N', then DLF is an output argument and on exit contains the (n-1) multipliers that define the matrix L from the LU factorization of A.

DF Input and output parameter.

DF is REAL

DF is an array, dimension (N). If FACT = 'F', then DF is an input argument and on entry contains the n diagonal elements of the upper triangular matrix U from the LU factorization of A.

If FACT = 'N', then DF is an output argument and on exit contains the n diagonal elements of the upper triangular matrix U from the LU factorization of A.

DUF Input and output parameter.

DUF is REAL

DUF is an array, dimension (N-1). If FACT = 'F', then DUF is an input argument and on entry contains the (n-1) elements of the first superdiagonal of U.

If FACT = 'N', then DUF is an output argument and on exit contains the (n-1) elements of the first superdiagonal of U.

DU2 Input and output parameter.

DU2 is REAL

DU2 is an array, dimension (N-2). If FACT = 'F', then DU2 is an input argument and on entry contains the (n-2) elements of the second superdiagonal of U.

If FACT = 'N', then DU2 is an output argument and on exit contains the (n-2) elements of the second superdiagonal of U.

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains the pivot indices from the LU factorization of A as computed by SGTTRF.

If FACT = 'N', then IPIV is an output argument and on exit contains the pivot indices from the LU factorization of A; row i of the matrix was interchanged with row IPIV(i). IPIV(i) will always be either i or i+1; IPIV(i) = i indicates a row interchange was not required.

B Input parameter.

B is REAL

B is an array, dimension (LDB, NRHS). The N-by-NRHS right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is REAL

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

The estimate of the reciprocal condition number of the matrix A. If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: U(i,i) is exactly zero. The factorization has not been completed unless i = N, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+1: U is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *cgtsvx*, *dgtsvx* and *zgtsvx*. It also exists with a native C interface as *LAPACKE_sgtsvx*.

4.6.108 sgttrs

sgttrs solves one of the systems of equations

```
A*X = B  or  A**T*X = B,
```

with a tridiagonal matrix A using the LU factorization computed by SGTTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgttrs(TRANS, N, NRHS, DL, D, DU, DU2, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void sgttrs_(const char *trans, const armpl_int_t *n, const armpl_int_t *nrhs,
             const float *dl, const float *d, const float *du,
             const float *du2, const armpl_int_t *ipiv, float *b,
             const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations. = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^T * X = B (Conjugate transpose = Transpose)

N Input parameter.

N is INTEGER

The order of the matrix A.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

DL Input parameter.

DL is REAL

DL is an array, dimension (N-1). The (n-1) multipliers that define the matrix L from the LU factorization of A.

D Input parameter.

D is REAL

D is an array, dimension (N). The n diagonal elements of the upper triangular matrix U from the LU factorization of A.

DU Input parameter.

DU is REAL

DU is an array, dimension (N-1). The (n-1) elements of the first super-diagonal of U.

DU2 Input parameter.

DU2 is REAL

DU2 is an array, dimension (N-2). The (n-2) elements of the second super-diagonal of U.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices; for 1 <= i <= n, row i of the matrix was interchanged with row IPIV(i). IPIV(i) will always be either i or i+1; IPIV(i) = i indicates a row interchange was not required.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the matrix of right hand side vectors B. On exit, B is overwritten by the solution vectors X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgttrs*, *dgttrs* and *zgttrs*. It also exists with a native C interface as *LAPACKE_sgttrs*.

4.6.109 spbsv

spbsv computes the solution to a real system of linear equations

```
A * X = B,
```

where A is an N-by-N symmetric positive definite band matrix and X and B are N-by-NRHS matrices.

The Cholesky decomposition is used to factor A as

```
A = U**T * U,  if UPLO = 'U', or
A = L * L**T,  if UPLO = 'L',
```

where U is an upper triangular band matrix, and L is a lower triangular band matrix, with the same number of superdiagonals or subdiagonals as A. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine spbsv(UPLO, N, KD, NRHS, AB, LDAB, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void spbsv_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
            const armpl_int_t *nrhs, float *ab, const armpl_int_t *ldab,
            float *b, const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AB Input and output parameter.

AB is REAL

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(KD+1+i-j,j) = A(i,j) for max(1,j-KD)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(N,j+KD). See below for further details.

On exit, if INFO = 0, the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T of the band matrix A, in the same storage format as A.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

Related Information

For this routine in other precisions, please see *cpbsv*, *dpbsv* and *zpbsv*. It also exists with a native C interface as *LAPACKE_spbsv*.

4.6.110 spbsvx

spbsvx uses the Cholesky factorization A = U^T *U or A = L*L^T to compute the solution to a real system of linear equations

```
A * X = B,
```

where A is an N-by-N symmetric positive definite band matrix and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed:
```

1. If FACT = 'E', real scaling factors are computed to equilibrate

```
 the system:
   diag(S) * A * diag(S) * inv(diag(S)) * X = diag(S) * B
Whether or not the system will be equilibrated depends on the
scaling of the matrix A, but if equilibration is used, A is
overwritten by diag(S)*A*diag(S) and B by diag(S)*B.
```

2. If FACT = 'N' or 'E', the Cholesky decomposition is used to

```
 factor the matrix A (after equilibration if FACT = 'E') as
   A = U**T * U,  if UPLO = 'U', or
   A = L * L**T,  if UPLO = 'L',
where U is an upper triangular band matrix, and L is a lower
triangular band matrix.
```

3. If the leading i-by-i principal minor is not positive definite,

```
 then the routine returns with INFO = i. Otherwise, the factored
form of A is used to estimate the condition number of the matrix
A.  If the reciprocal of the condition number is less than machine
precision, INFO = N+1 is returned as a warning, but the routine
still goes on to solve for X and compute error bounds as
described below.
```

4. The system of equations is solved for X using the factored form

```
of A.
```

5. Iterative refinement is applied to improve the computed solution

```
 matrix and calculate error bounds and backward error estimates
for it.
```

6. If equilibration was used, the matrix X is premultiplied by

```
 diag(S) so that it solves the original system before
equilibration.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine spbsvx(FACT, UPLO, N, KD, NRHS, AB, LDAB, AFB, LDAFB, EQUED, S, B,
                  LDB, X, LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"
```

(continues on next page)

(continued from previous page)

```
void spbsvx_(const char *fact, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *kd, const armpl_int_t *nrhs, float *ab,
             const armpl_int_t *ldab, float *afb, const armpl_int_t *ldafb,
             char *equed, float *s, float *b, const armpl_int_t *ldb,
             float *x, const armpl_int_t *ldx, float *rcond, float *ferr,
             float *berr, float *work, armpl_int_t *iwork, armpl_int_t *info,
             ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AFB contains the factored form of A. If EQUED = 'Y', the matrix A has been equilibrated with scaling factors given by S. AB and AFB will not be modified. = 'N': The matrix A will be copied to AFB and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AFB and factored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right-hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AB Input and output parameter.

AB is REAL

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first KD+1 rows of the array, except if FACT = 'F' and EQUED = 'Y', then A must contain the equilibrated matrix diag(S)*A*diag(S). The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(KD+1+i-j,j) = A(i,j) for max(1,j-KD)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(N,j+KD). See below for further details.

On exit, if FACT = 'E' and EQUED = 'Y', A is overwritten by diag(S)*A*diag(S).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array A. LDAB >= KD+1.

AFB Input and output parameter.

AFB is REAL

AFB is an array, dimension (LDAFB, N). If FACT = 'F', then AFB is an input argument and on entry contains the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T of the band matrix A, in the same storage format as A (see AB). If EQUED = 'Y', then AFB is the factored form of the equilibrated matrix A.

If FACT = 'N', then AFB is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T .

If FACT = 'E', then AFB is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T of the equilibrated matrix A (see the description of A for the form of the equilibrated matrix).

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AFB. LDAFB >= KD+1.

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'Y': Equilibration was done, i.e., A has been replaced by diag(S) * A * diag(S). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

S Input and output parameter.

S is REAL

S is an array, dimension (N). The scale factors for A; not accessed if EQUED = 'N'. S is an input argument if FACT = 'F'; otherwise, S is an output argument. If FACT = 'F' and EQUED = 'Y', each element of S must be positive.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if EQUED = 'N', B is not modified; if EQUED = 'Y', B is overwritten by diag(S) * B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is REAL

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X to the original system of equations. Note that if EQUED = 'Y', A and B are modified on exit, and the solution to the equilibrated system is inv(diag(S))*X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

The estimate of the reciprocal condition number of the matrix A after equilibration (if done). If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed. RCOND = 0 is returned. = N+1: U is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *cpbsvx*, *dpbsvx* and *zpbsvx*. It also exists with a native C interface as *LAPACKE_spbsvx*.

4.6.111 spbtrs

spbtrs solves a system of linear equations A*X = B with a symmetric positive definite band matrix A using the Cholesky factorization A = U^T *U or A = L*L^T computed by SPBTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine spbtrs(UPLO, N, KD, NRHS, AB, LDAB, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void spbtrs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
             const armpl_int_t *nrhs, const float *ab,
             const armpl_int_t *ldab, float *b, const armpl_int_t *ldb,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangular factor stored in AB; = 'L': Lower triangular factor stored in AB.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AB Input parameter.

AB is REAL

AB is an array, dimension (LDAB, N). The triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T of the band matrix A, stored in the first KD+1 rows of the array. The j-th column of U or L is stored in the j-th column of the array AB as follows: if UPLO ='U', AB(kd+1+i-j,j) = U(i,j) for max(1,j-kd)<=i<=j; if UPLO ='L', AB(1+i-j,j) = L(i,j) for j<=i<=min(n,j+kd).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cpbtrs*, *dpbtrs* and *zpbtrs*. It also exists with a native C interface as *LAPACKE_spbtrs*.

4.6.112 spftrs

spftrs solves a system of linear equations A*X = B with a symmetric positive definite matrix A using the Cholesky factorization A = U^T *U or A = L*L^T computed by SPFTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine spftrs(TRANSR, UPLO, N, NRHS, A, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void spftrs_(const char *transr, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *nrhs, const float *a, float *b,
             const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': The Normal TRANSR of RFP A is stored; = 'T': The Transpose TRANSR of RFP A is stored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of RFP A is stored; = 'L': Lower triangle of RFP A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is REAL

A is an array, dimension (N*(N+1)/2). The triangular factor U or L from the Cholesky factorization of RFP A = U^H *U or RFP A = L*L^T , as computed by SPFTRF. See note below for more details about RFP A.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cpftrs*, *dpftrs* and *zpftrs*. It also exists with a native C interface as *LAPACKE_spftrs*.

4.6.113 sposv

sposv computes the solution to a real system of linear equations

```
A * X = B,
```

where A is an N-by-N symmetric positive definite matrix and X and B are N-by-NRHS matrices.

The Cholesky decomposition is used to factor A as

```
A = U**T* U,  if UPLO = 'U', or
A = L * L**T,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is a lower triangular matrix. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine sposv(UPLO, N, NRHS, A, LDA, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void sposv_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
            float *a, const armpl_int_t *lda, float *b,
            const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular

part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T .

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

Related Information

For this routine in other precisions, please see *cposv*, *dposv* and *zposv*. It also exists with a native C interface as *LAPACKE_sposv*.

4.6.114 sposvx

sposvx uses the Cholesky factorization A = U^T *U or A = L*L^T to compute the solution to a real system of linear equations

```
A * X = B,
```

where A is an N-by-N symmetric positive definite matrix and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed:
```

1. If FACT = 'E', real scaling factors are computed to equilibrate

```
 the system:
   diag(S) * A * diag(S) * inv(diag(S)) * X = diag(S) * B
Whether or not the system will be equilibrated depends on the
scaling of the matrix A, but if equilibration is used, A is
overwritten by diag(S)*A*diag(S) and B by diag(S)*B.
```

2. If FACT = 'N' or 'E', the Cholesky decomposition is used to

```
 factor the matrix A (after equilibration if FACT = 'E') as
   A = U**T* U,  if UPLO = 'U', or
   A = L * L**T,  if UPLO = 'L',
where U is an upper triangular matrix and L is a lower triangular
matrix.
```

3. If the leading i-by-i principal minor is not positive definite,

```
 then the routine returns with INFO = i. Otherwise, the factored
form of A is used to estimate the condition number of the matrix
A.  If the reciprocal of the condition number is less than machine
precision, INFO = N+1 is returned as a warning, but the routine
still goes on to solve for X and compute error bounds as
described below.
```

4. The system of equations is solved for X using the factored form

```
of A.
```

5. Iterative refinement is applied to improve the computed solution

```
 matrix and calculate error bounds and backward error estimates
for it.
```

6. If equilibration was used, the matrix X is premultiplied by

```
 diag(S) so that it solves the original system before
equilibration.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine sposvx(FACT, UPLO, N, NRHS, A, LDA, AF, LDAF, EQUED, S, B, LDB, X,
                  LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sposvx_(const char *fact, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *nrhs, float *a, const armpl_int_t *lda,
             float *af, const armpl_int_t *ldaf, char *equed, float *s,
             float *b, const armpl_int_t *ldb, float *x,
             const armpl_int_t *ldx, float *rcond, float *ferr, float *berr,
             float *work, armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AF contains the factored form of A. If EQUED = 'Y', the matrix A has been equilibrated with scaling factors given by S. A and AF will not be modified. = 'N': The matrix A will be copied to AF and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AF and factored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A, except if FACT = 'F' and EQUED = 'Y', then A must contain the equilibrated matrix diag(S)*A*diag(S). If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. A is not modified if FACT = 'F' or 'N', or if FACT = 'E' and EQUED = 'N' on exit.

On exit, if FACT = 'E' and EQUED = 'Y', A is overwritten by diag(S)*A*diag(S).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input and output parameter.

AF is REAL

AF is an array, dimension (LDAF, N). If FACT = 'F', then AF is an input argument and on entry contains the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , in the same storage format as A. If EQUED .ne. 'N', then AF is the factored form of the equilibrated matrix diag(S)*A*diag(S).

If FACT = 'N', then AF is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T of the original matrix A.

If FACT = 'E', then AF is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T of the equilibrated matrix A (see the description of A for the form of the equilibrated matrix).

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'Y': Equilibration was done, i.e., A has been replaced by diag(S) * A * diag(S). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

S Input and output parameter.

S is REAL

S is an array, dimension (N). The scale factors for A; not accessed if EQUED = 'N'. S is an input argument if FACT = 'F'; otherwise, S is an output argument. If FACT = 'F' and EQUED = 'Y', each element of S must be positive.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if EQUED = 'N', B is not modified; if EQUED = 'Y', B is overwritten by diag(S) * B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is REAL

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X to the original system of equations. Note that if EQUED = 'Y', A and B are modified on exit, and the solution to the equilibrated system is inv(diag(S))*X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

The estimate of the reciprocal condition number of the matrix A after equilibration (if done). If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed. RCOND = 0 is returned. = N+1: U is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *cposvx*, *dposvx* and *zposvx*. It also exists with a native C interface as *LAPACKE_sposvx*.

4.6.115 sposvxx

```
SPOSVXX uses the Cholesky factorization A = U**T*U or A = L*L**T
   to compute the solution to a real system of linear equations
   A * X = B, where A is an N-by-N symmetric positive definite matrix
   and X and B are N-by-NRHS matrices.

   If requested, both normwise and maximum componentwise error bounds
   are returned. SPOSVXX will return a solution with a tiny
   guaranteed error (O(eps) where eps is the working machine
   precision) unless the matrix is very ill-conditioned, in which
   case a warning is returned. Relevant condition numbers also are
   calculated and returned.

   SPOSVXX accepts user-provided factorizations and equilibration
   factors; see the definitions of the FACT and EQUED options.
   Solving with refinement and using a factorization from a previous
   SPOSVXX call will also produce a solution with either O(eps)
   errors or warnings, but we cannot make that claim for general
   user-provided factorizations and equilibration factors if they
   differ from what SPOSVXX would itself produce.

                                                                    The␣
↪following steps are performed:

   1. If FACT = 'E', real scaling factors are computed to equilibrate
   the system:

     diag(S)*A*diag(S)      *inv(diag(S))*X = diag(S)*B

   Whether or not the system will be equilibrated depends on the
   scaling of the matrix A, but if equilibration is used, A is
   overwritten by diag(S)*A*diag(S) and B by diag(S)*B.

   2. If FACT = 'N' or 'E', the Cholesky decomposition is used to
   factor the matrix A (after equilibration if FACT = 'E') as
      A = U**T* U,  if UPLO = 'U', or
      A = L * L**T,  if UPLO = 'L',
   where U is an upper triangular matrix and L is a lower triangular
   matrix.

   3. If the leading i-by-i principal minor is not positive definite,
   then the routine returns with INFO = i. Otherwise, the factored
   form of A is used to estimate the condition number of the matrix
   A (see argument RCOND).  If the reciprocal of the condition number
   is less than machine precision, the routine still goes on to solve
   for X and compute error bounds as described below.

   4. The system of equations is solved for X using the factored form
   of A.

   5. By default (unless PARAMS(LA_LINRX_ITREF_I) is set to zero),
   the routine will use iterative refinement to try to get a small
   error and error bounds.  Refinement calculates the residual to at
   least twice the working precision.
```

(continues on next page)

(continued from previous page)

```
   6. If equilibration was used, the matrix X is premultiplied by
   diag(S) so that it solves the original system before
   equilibration.                          Some optional parameters are bundled␣
↪in the PARAMS array.  These
    settings determine how refinement is performed, but often the
    defaults are acceptable.  If the defaults are acceptable, users
    can pass NPARAMS = 0 which prevents the source code from accessing
    the PARAMS argument.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine sposvxx(FACT, UPLO, N, NRHS, A, LDA, AF, LDAF, EQUED, S, B, LDB, X,
                   LDX, RCOND, RPVGRW, BERR, N_ERR_BNDS, ERR_BNDS_NORM,
                   ERR_BNDS_COMP, NPARAMS, PARAMS, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sposvxx_(const char *fact, const char *uplo, const armpl_int_t *n,
              const armpl_int_t *nrhs, float *a, const armpl_int_t *lda,
              float *af, const armpl_int_t *ldaf, char *equed, float *s,
              float *b, const armpl_int_t *ldb, float *x,
              const armpl_int_t *ldx, float *rcond, float *rpvgrw,
              float *berr, const armpl_int_t *n_err_bnds,
              float *err_bnds_norm, float *err_bnds_comp,
              const armpl_int_t *nparams, float *params, float *work,
              armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AF contains the factored form of A. If EQUED is not 'N', the matrix A has been equilibrated with scaling factors given by S. A and AF are not modified. = 'N': The matrix A will be copied to AF and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AF and factored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A, except if FACT = 'F' and EQUED = 'Y', then A must contain the equilibrated matrix diag(S)*A*diag(S). If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. A is not modified if FACT = 'F' or 'N', or if FACT = 'E' and EQUED = 'N' on exit.

On exit, if FACT = 'E' and EQUED = 'Y', A is overwritten by diag(S)*A*diag(S).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input and output parameter.

AF is REAL

AF is an array, dimension (LDAF, N). If FACT = 'F', then AF is an input argument and on entry contains the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , in the same storage format as A. If EQUED .ne. 'N', then AF is the factored form of the equilibrated matrix diag(S)*A*diag(S).

If FACT = 'N', then AF is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T of the original matrix A.

If FACT = 'E', then AF is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T of the equilibrated matrix A (see the description of A for the form of the equilibrated matrix).

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'Y': Both row and column equilibration, i.e., A has been replaced by diag(S) * A * diag(S). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

S Input and output parameter.

S is REAL

S is an array, dimension (N). The row scale factors for A. If EQUED = 'Y', A is multiplied on the left and right by diag(S). S is an input argument if FACT = 'F'; otherwise, S is an output argument. If FACT = 'F' and EQUED = 'Y', each element of S must be positive. If S is output, each element of S is a power of the radix. If S is input, each element of S should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if EQUED = 'N', B is not modified; if EQUED = 'Y', B is overwritten by diag(S)*B;

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is REAL

X is an array, dimension (LDX, NRHS). If INFO = 0, the N-by-NRHS solution matrix X to the original system of equations. Note that A and B are modified on exit if EQUED .ne. 'N', and the solution to the equilibrated system is inv(diag(S))*X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

RPVGRW Output parameter.

RPVGRW is REAL

Reciprocal pivot growth. On exit, this contains the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If this is much less than 1, then the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, estimated condition numbers, and error bounds could be unreliable. If factorization fails with 0<INFO<=N, then this contains the reciprocal pivot growth factor for the leading INFO columns of A.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). Componentwise relative backward error. This is the componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

N_ERR_BNDS Input parameter.

N_ERR_BNDS is INTEGER

Number of error bounds to return for each right hand side and each type (normwise or componentwise). See ERR_BNDS_NORM and ERR_BNDS_COMP below.

ERR_BNDS_NORM Output parameter.

ERR_BNDS_NORM is REAL

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 “Guaranteed” error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch(‘Epsilon’). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch(‘Epsilon’) to determine if the error estimate is “guaranteed”. These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Output parameter.

ERR_BNDS_COMP is REAL

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:, N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 “Trust/don’t trust” boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch(‘Epsilon’).

err = 2 “Guaranteed” error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch(‘Epsilon’). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch(‘Epsilon’) to determine if the error estimate is “guaranteed”. These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

NPARAMS Input parameter.

NPARAMS is INTEGER

Specifies the number of parameters set in PARAMS. If .LE. 0, the PARAMS array is never referenced and default values are used.

PARAMS Input and output parameter.

PARAMS is REAL

PARAMS is an array, dimension NPARAMS. Specifies algorithm parameters. If an entry is .LT. 0.0, then that entry will be filled with default value used for that parameter. Only positions up to NPARAMS are accessed; defaults are used for higher-numbered parameters.

PARAMS(LA_LINRX_ITREF_I = 1) : Whether to perform iterative refinement or not. Default: 1.0 = 0.0 : No refinement is performed, and no error bounds are computed. = 1.0 : Use the double-precision refinement algorithm, possibly with doubled-single computations if the compilation environment does not support DOUBLE PRECISION. (other values are reserved for future use)

PARAMS(LA_LINRX_ITHRESH_I = 2) : Maximum number of residual computations allowed for refinement. Default: 10 Aggressive: Set to 100 to permit convergence using approximate factorizations or

factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in err_bnds_norm and err_bnds_comp may no longer be trustworthy.

PARAMS(LA_LINRX_CWISE_I = 3) : Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Default: 1.0 (attempt componentwise convergence)

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (4*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. The solution to every right-hand side is guaranteed. < 0: If INFO = -i, the i-th argument had an illegal value > 0 and <= N: U(INFO, INFO) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+J: The solution corresponding to the Jth right-hand side is not guaranteed. The solutions corresponding to other right- hand sides K with K > J may not be guaranteed as well, but only the first such right-hand side is reported. If a small componentwise error is not requested (PARAMS(3) = 0.0) then the Jth right-hand side is the first with a normwise error bound that is not guaranteed (the smallest J such that ERR_BNDS_NORM(J,1) = 0.0). By default (PARAMS(3) = 1.0) the Jth right-hand side is the first with either a normwise or componentwise error bound that is not guaranteed (the smallest J such that either ERR_BNDS_NORM(J,1) = 0.0 or ERR_BNDS_COMP(J,1) = 0.0). See the definition of ERR_BNDS_NORM(:,1) and ERR_BNDS_COMP(:,1). To get information about all of the right-hand sides check ERR_BNDS_NORM or ERR_BNDS_COMP.

Related Information

For this routine in other precisions, please see *cposvxx*, *dposvxx* and *zposvxx*. It also exists with a native C interface as *LAPACKE_sposvxx*.

4.6.116 spotrs

spotrs solves a system of linear equations A*X = B with a symmetric positive definite matrix A using the Cholesky factorization $A = U^T * U$ or $A = L * L^T$ computed by SPOTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine spotrs(UPLO, N, NRHS, A, LDA, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void spotrs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const float *a, const armpl_int_t *lda, float *b,
             const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). The triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , as computed by SPOTRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cpotrs*, *dpotrs* and *zpotrs*. It also exists with a native C interface as *LAPACKE_spotrs*.

4.6.117 sppsv

sppsv computes the solution to a real system of linear equations

```
A * X = B,
```

where A is an N-by-N symmetric positive definite matrix stored in packed format and X and B are N-by-NRHS matrices.

The Cholesky decomposition is used to factor A as

```
A = U**T* U,  if UPLO = 'U', or
A = L * L**T,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is a lower triangular matrix. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine sppsv(UPLO, N, NRHS, AP, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void sppsv_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
            float *ap, float *b, const armpl_int_t *ldb, armpl_int_t *info,
            ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AP Input and output parameter.

AP is REAL

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n. See below for further details.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , in the same storage format as A.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

Related Information

For this routine in other precisions, please see *cppsv*, *dppsv* and *zppsv*. It also exists with a native C interface as *LAPACKE_sppsv*.

4.6.118 sppsvx

sppsvx uses the Cholesky factorization A = U^T *U or A = L*L^T to compute the solution to a real system of linear equations

```
A * X = B,
```

where A is an N-by-N symmetric positive definite matrix stored in packed format and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed:
```

1. If FACT = 'E', real scaling factors are computed to equilibrate

```
 the system:
   diag(S) * A * diag(S) * inv(diag(S)) * X = diag(S) * B
Whether or not the system will be equilibrated depends on the
scaling of the matrix A, but if equilibration is used, A is
overwritten by diag(S)*A*diag(S) and B by diag(S)*B.
```

2. If FACT = 'N' or 'E', the Cholesky decomposition is used to

```
 factor the matrix A (after equilibration if FACT = 'E') as
   A = U**T* U,  if UPLO = 'U', or
   A = L * L**T,  if UPLO = 'L',
where U is an upper triangular matrix and L is a lower triangular
matrix.
```

3. If the leading i-by-i principal minor is not positive definite,

```
 then the routine returns with INFO = i. Otherwise, the factored
form of A is used to estimate the condition number of the matrix
A.  If the reciprocal of the condition number is less than machine
precision, INFO = N+1 is returned as a warning, but the routine
still goes on to solve for X and compute error bounds as
described below.
```

4. The system of equations is solved for X using the factored form

```
of A.
```

5. Iterative refinement is applied to improve the computed solution

```
 matrix and calculate error bounds and backward error estimates
for it.
```

6. If equilibration was used, the matrix X is premultiplied by

```
 diag(S) so that it solves the original system before
equilibration.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine sppsvx(FACT, UPLO, N, NRHS, AP, AFP, EQUED, S, B, LDB, X, LDX,
                  RCOND, FERR, BERR, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sppsvx_(const char *fact, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *nrhs, float *ap, float *afp, char *equed,
             float *s, float *b, const armpl_int_t *ldb, float *x,
             const armpl_int_t *ldx, float *rcond, float *ferr, float *berr,
             float *work, armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AFP contains the factored form of A. If EQUED = 'Y', the matrix A has been equilibrated with scaling factors given by S. AP and AFP will not be modified. = 'N': The matrix A will be copied to AFP and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AFP and factored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AP Input and output parameter.

AP is REAL

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array, except if FACT = 'F' and EQUED = 'Y', then A must contain the equilibrated matrix diag(S)*A*diag(S). The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n. See below for further details. A is not modified if FACT = 'F' or 'N', or if FACT = 'E' and EQUED = 'N' on exit.

On exit, if FACT = 'E' and EQUED = 'Y', A is overwritten by diag(S)*A*diag(S).

AFP Input and output parameter.

AFP is REAL

AFP is an array, dimension (N*(N+1)/2). If FACT = 'F', then AFP is an input argument and on entry contains the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , in the same storage format as A. If EQUED .ne. 'N', then AFP is the factored form of the equilibrated matrix A.

If FACT = 'N', then AFP is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization A = U^T * U or A = L * L^T of the original matrix A.

If FACT = 'E', then AFP is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization A = U^T * U or A = L * L^T of the equilibrated matrix A (see the description of AP for the form of the equilibrated matrix).

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'Y': Equilibration was done, i.e., A has been replaced by diag(S) * A * diag(S). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

S Input and output parameter.

S is REAL

S is an array, dimension (N). The scale factors for A; not accessed if EQUED = 'N'. S is an input argument if FACT = 'F'; otherwise, S is an output argument. If FACT = 'F' and EQUED = 'Y', each element of S must be positive.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if EQUED = 'N', B is not modified; if EQUED = 'Y', B is overwritten by diag(S) * B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is REAL

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X to the original system of equations. Note that if EQUED = 'Y', A and B are modified on exit, and the solution to the equilibrated system is inv(diag(S))*X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

The estimate of the reciprocal condition number of the matrix A after equilibration (if done). If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an

estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed. RCOND = 0 is returned. = N+1: U is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *cppsvx*, *dppsvx* and *zppsvx*. It also exists with a native C interface as *LAPACKE_sppsvx*.

4.6.119 spptrs

spptrs solves a system of linear equations A*X = B with a symmetric positive definite matrix A in packed storage using the Cholesky factorization $A = U^T * U$ or $A = L * L^T$ computed by SPPTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine spptrs(UPLO, N, NRHS, AP, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void spptrs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const float *ap, float *b, const armpl_int_t *ldb,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AP Input parameter.

AP is REAL

AP is an array, dimension (N*(N+1)/2). The triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , packed columnwise in a linear array. The j-th column of U or L is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = U(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = L(i,j) for j<=i<=n.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cpptrs*, *dpptrs* and *zpptrs*. It also exists with a native C interface as *LAPACKE_spptrs*.

4.6.120 sptsv

sptsv computes the solution to a real system of linear equations A*X = B, where A is an N-by-N symmetric positive definite tridiagonal matrix, and X and B are N-by-NRHS matrices.

A is factored as A = L*D*L^T , and the factored form of A is then used to solve the system of equations.

Syntax

Fortran specification:

```
use armpl_library

subroutine sptsv(N, NRHS, D, E, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void sptsv_(const armpl_int_t *n, const armpl_int_t *nrhs, float *d, float *e,
            float *b, const armpl_int_t *ldb, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry, the n diagonal elements of the tridiagonal matrix A. On exit, the n diagonal elements of the diagonal matrix D from the factorization A = L*D*L^T .

E Input and output parameter.

E is REAL

E is an array, dimension (N-1). On entry, the (n-1) subdiagonal elements of the tridiagonal matrix A. On exit, the (n-1) subdiagonal elements of the unit bidiagonal factor L from the L*D*L^T factorization of A. (E can also be regarded as the superdiagonal of the unit bidiagonal factor U from the U^T *D*U factorization of A.)

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i is not positive definite, and the solution has not been computed. The factorization has not been completed unless i = N.

Related Information

For this routine in other precisions, please see *cptsv*, *dptsv* and *zptsv*. It also exists with a native C interface as *LAPACKE_sptsv*.

4.6.121 sptsvx

sptsvx uses the factorization A = L*D*L^T to compute the solution to a real system of linear equations A*X = B, where A is an N-by-N symmetric positive definite tridiagonal matrix and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed:
```

1. If FACT = 'N', the matrix A is factored as A = L*D*L^T , where L

```
 is a unit lower bidiagonal matrix and D is diagonal.  The
factorization can also be regarded as having the form
A = U**T*D*U.
```

2. If the leading i-by-i principal minor is not positive definite,

```
 then the routine returns with INFO = i. Otherwise, the factored
form of A is used to estimate the condition number of the matrix
A.  If the reciprocal of the condition number is less than machine
precision, INFO = N+1 is returned as a warning, but the routine
still goes on to solve for X and compute error bounds as
described below.
```

3. The system of equations is solved for X using the factored form

```
of A.
```

4. Iterative refinement is applied to improve the computed solution

```
 matrix and calculate error bounds and backward error estimates
for it.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine sptsvx(FACT, N, NRHS, D, E, DF, EF, B, LDB, X, LDX, RCOND, FERR,
                  BERR, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void sptsvx_(const char *fact, const armpl_int_t *n, const armpl_int_t *nrhs,
             const float *d, const float *e, float *df, float *ef,
             const float *b, const armpl_int_t *ldb, float *x,
             const armpl_int_t *ldx, float *rcond, float *ferr, float *berr,
             float *work, armpl_int_t *info, ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of A has been supplied on entry. = 'F': On entry, DF and EF contain the factored form of A. D, E, DF, and EF will not be modified. = 'N': The matrix A will be copied to DF and EF and factored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

D Input parameter.

D is REAL

D is an array, dimension (N). The n diagonal elements of the tridiagonal matrix A.

E Input parameter.

E is REAL

E is an array, dimension (N-1). The (n-1) subdiagonal elements of the tridiagonal matrix A.

DF Input and output parameter.

DF is REAL

DF is an array, dimension (N). If FACT = 'F', then DF is an input argument and on entry contains the n diagonal elements of the diagonal matrix D from the $L*D*L^T$ factorization of A. If FACT = 'N', then DF is an output argument and on exit contains the n diagonal elements of the diagonal matrix D from the $L*D*L^T$ factorization of A.

EF Input and output parameter.

EF is REAL

EF is an array, dimension (N-1). If FACT = 'F', then EF is an input argument and on entry contains the (n-1) subdiagonal elements of the unit bidiagonal factor L from the $L*D*L^T$ factorization of A. If FACT = 'N', then EF is an output argument and on exit contains the (n-1) subdiagonal elements of the unit bidiagonal factor L from the $L*D*L^T$ factorization of A.

B Input parameter.

B is REAL

B is an array, dimension (LDB, NRHS). The N-by-NRHS right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is REAL

X is an array, dimension (LDX, NRHS). If INFO = 0 of INFO = N+1, the N-by-NRHS solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

The reciprocal condition number of the matrix A. If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j).

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed. RCOND = 0 is returned. = N+1: U is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *cptsvx*, *dptsvx* and *zptsvx*. It also exists with a native C interface as *LAPACKE_sptsvx*.

4.6.122 spttrs

spttrs solves a tridiagonal system of the form

```
A * X = B
```

using the L*D*L^T factorization of A computed by SPTTRF. D is a diagonal matrix specified in the vector D, L is a unit bidiagonal matrix whose subdiagonal is specified in the vector E, and X and B are N by NRHS matrices.

Syntax

Fortran specification:

```
use armpl_library

subroutine spttrs(N, NRHS, D, E, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void spttrs_(const armpl_int_t *n, const armpl_int_t *nrhs, const float *d,
             const float *e, float *b, const armpl_int_t *ldb,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the tridiagonal matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

D Input parameter.

D is REAL

D is an array, dimension (N). The n diagonal elements of the diagonal matrix D from the L*D*L^T factorization of A.

E Input parameter.

E is REAL

E is an array, dimension (N-1). The (n-1) subdiagonal elements of the unit bidiagonal factor L from the L*D*L^T factorization of A. E can also be regarded as the superdiagonal of the unit bidiagonal factor U from the factorization A = U^T *D*U.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the right hand side vectors B for the system of linear equations. On exit, the solution vectors, X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cpttrs*, *dpttrs* and *zpttrs*. It also exists with a native C interface as *LAPACKE_spttrs*.

4.6.123 sspsv

sspsv computes the solution to a real system of linear equations

```
A * X = B,
```

where A is an N-by-N symmetric matrix stored in packed format and X and B are N-by-NRHS matrices.

The diagonal pivoting method is used to factor A as

```
 A = U * D * U**T,  if UPLO = 'U', or
A = L * D * L**T,  if UPLO = 'L',
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine sspsv(UPLO, N, NRHS, AP, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void sspsv_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
            float *ap, armpl_int_t *ipiv, float *b, const armpl_int_t *ldb,
            armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AP Input and output parameter.

AP is REAL

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n. See below for further details.

On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by SSPTRF, stored as a packed triangular matrix in the same storage format as A.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D, as determined by SSPTRF. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged, and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, so the solution could not be computed.

Related Information

For this routine in other precisions, please see *cspsv*, *dspsv* and *zspsv*. It also exists with a native C interface as *LAPACKE_sspsv*.

4.6.124 sspsvx

sspsvx uses the diagonal pivoting factorization A = U*D*U^T or A = L*D*L^T to compute the solution to a real system of linear equations A * X = B, where A is an N-by-N symmetric matrix stored in packed format and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed:
```

1. If FACT = 'N', the diagonal pivoting method is used to factor A as

```
   A = U * D * U**T,  if UPLO = 'U', or
   A = L * D * L**T,  if UPLO = 'L',
where U (or L) is a product of permutation and unit upper (lower)
triangular matrices and D is symmetric and block diagonal with
1-by-1 and 2-by-2 diagonal blocks.
```

2. If some D(i,i)=0, so that D is exactly singular, then the routine

```
 returns with INFO = i. Otherwise, the factored form of A is used
to estimate the condition number of the matrix A.  If the
reciprocal of the condition number is less than machine precision,
INFO = N+1 is returned as a warning, but the routine still goes on
to solve for X and compute error bounds as described below.
```

3. The system of equations is solved for X using the factored form

```
of A.
```

4. Iterative refinement is applied to improve the computed solution

```
 matrix and calculate error bounds and backward error estimates
for it.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine sspsvx(FACT, UPLO, N, NRHS, AP, AFP, IPIV, B, LDB, X, LDX, RCOND,
                  FERR, BERR, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sspsvx_(const char *fact, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *nrhs, const float *ap, float *afp,
             armpl_int_t *ipiv, const float *b, const armpl_int_t *ldb,
             float *x, const armpl_int_t *ldx, float *rcond, float *ferr,
             float *berr, float *work, armpl_int_t *iwork, armpl_int_t *info,
             ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of A has been supplied on entry. = 'F': On entry, AFP and IPIV contain the factored form of A. AP, AFP and IPIV will not be modified. = 'N': The matrix A will be copied to AFP and factored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AP Input parameter.

AP is REAL

AP is an array, dimension (N*(N+1)/2). The upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n. See below for further details.

AFP Input and output parameter.

AFP is REAL

AFP is an array, dimension (N*(N+1)/2). If FACT = 'F', then AFP is an input argument and on entry contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by SSPTRF, stored as a packed triangular matrix in the same storage format as A.

If FACT = 'N', then AFP is an output argument and on exit contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by SSPTRF, stored as a packed triangular matrix in the same storage format as A.

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains details of the interchanges and the block structure of D, as determined by SSPTRF. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

If FACT = 'N', then IPIV is an output argument and on exit contains details of the interchanges and the block structure of D, as determined by SSPTRF.

B Input parameter.

B is REAL

B is an array, dimension (LDB, NRHS). The N-by-NRHS right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is REAL

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

The estimate of the reciprocal condition number of the matrix A. If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: D(i,i) is exactly zero. The factorization has been completed but the factor D is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+1: D is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *cspsvx*, *dspsvx* and *zspsvx*. It also exists with a native C interface as *LAPACKE_sspsvx*.

4.6.125 ssptrs

ssptrs solves a system of linear equations A*X = B with a real symmetric matrix A stored in packed format using the factorization A = U*D*U^T or A = L*D*L^T computed by SSPTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssptrs(UPLO, N, NRHS, AP, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void ssptrs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const float *ap, const armpl_int_t *ipiv, float *b,
             const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AP Input parameter.

AP is REAL

AP is an array, dimension (N*(N+1)/2). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by SSPTRF, stored as a packed triangular matrix.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by SSPTRF.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csptrs*, *dsptrs* and *zsptrs*. It also exists with a native C interface as *LAPACKE_ssptrs*.

4.6.126 ssysv

ssysv computes the solution to a real system of linear equations

```
A * X = B,
```

where A is an N-by-N symmetric matrix and X and B are N-by-NRHS matrices.

The diagonal pivoting method is used to factor A as

```
A = U * D * U**T,  if UPLO = 'U', or
A = L * D * L**T,  if UPLO = 'L',
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssysv(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ssysv_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
            float *a, const armpl_int_t *lda, armpl_int_t *ipiv, float *b,
            const armpl_int_t *ldb, float *work, const armpl_int_t *lwork,
            armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by SSYTRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D, as determined by SSYTRF. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged, and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= 1, and for best performance LWORK >= max(1,N*NB), where NB is the optimal blocksize for SSYTRF. for LWORK < N, TRS will be done with Level BLAS 2 for LWORK >= N, TRS will be done with Level BLAS 3

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, so the solution could not be computed.

Related Information

For this routine in other precisions, please see *csysv*, *dsysv* and *zsysv*. It also exists with a native C interface as *LAPACKE_ssysv*.

4.6.127 ssysv_aa

SSYSV computes the solution to a real system of linear equations

```
A * X = B,
```

where A is an N-by-N symmetric matrix and X and B are N-by-NRHS matrices.

Aasen's algorithm is used to factor A as

```
A = U * T * U**T,  if UPLO = 'U', or
A = L * T * L**T,  if UPLO = 'L',
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and T is symmetric tridiagonal. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssysv_aa(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ssysv_aa_(const char *uplo, const armpl_int_t *n,
               const armpl_int_t *nrhs, float *a, const armpl_int_t *lda,
```

(continues on next page)

(continued from previous page)

```
                armpl_int_t *ipiv, float *b, const armpl_int_t *ldb,
                float *work, const armpl_int_t *lwork, armpl_int_t *info,
                ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the tridiagonal matrix T and the multipliers used to obtain the factor U or L from the factorization A = U*T*U^T or A = L*T*L^T as computed by SSYTRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

On exit, it contains the details of the interchanges, i.e., the row and column k of A were interchanged with the row and column IPIV(k).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= MAX(1,2*N,3*N-2), and for the best performance, LWORK >= MAX(1,N*NB), where NB is the optimal blocksize for SSYTRF_AA.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, so the solution could not be computed.

Related Information

For this routine in other precisions, please see *csysv_aa*, *dsysv_aa* and *zsysv_aa*. It also exists with a native C interface as *LAPACKE_ssysv_aa*.

4.6.128 ssysv_rk

SSYSV_RK computes the solution to a real system of linear equations A * X = B, where A is an N-by-N symmetric matrix and X and B are N-by-NRHS matrices.

The bounded Bunch-Kaufman (rook) diagonal pivoting method is used to factor A as

```
A = P*U*D*(U**T)*(P**T),  if UPLO = 'U', or
A = P*L*D*(L**T)*(P**T),  if UPLO = 'L',
```

where U (or L) is unit upper (or lower) triangular matrix, U^T (or L^T) is the transpose of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

SSYTRF_RK is called to compute the factorization of a real symmetric matrix. The factored form of A is then used to solve the system of equations A * X = B by calling BLAS3 routine SSYTRS_3.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssysv_rk(UPLO, N, NRHS, A, LDA, E, IPIV, B, LDB, WORK, LWORK,
                    INFO)
```

C specification:

```
#include "armpl.h"

void ssysv_rk_(const char *uplo, const armpl_int_t *n,
               const armpl_int_t *nrhs, float *a, const armpl_int_t *lda,
               float *e, armpl_int_t *ipiv, float *b, const armpl_int_t *ldb,
               float *work, const armpl_int_t *lwork, armpl_int_t *info,
               ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U': the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced.

If UPLO = 'L': the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, diagonal of the block diagonal matrix D and factors U or L as computed by SSYTRF_RK: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

For more info see the description of DSYTRF_RK routine.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Output parameter.

E is REAL

E is an array, dimension (N). On exit, contains the output computed by the factorization routine DSYTRF_RK, i.e. the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) is set to 0; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) is set to 0.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is set to 0 in both UPLO = 'U' or UPLO = 'L' cases.

For more info see the description of DSYTRF_RK routine.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D, as determined by SSYTRF_RK.

For more info see the description of DSYTRF_RK routine.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)).. Work array used in the factorization stage. On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= 1. For best performance of factorization stage LWORK >= max(1,N*NB), where NB is the optimal blocksize for DSYTRF_RK.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array for factorization stage, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit

< 0: If INFO = -k, the k-th argument had an illegal value

> 0: If INFO = k, the matrix A is singular, because: If UPLO = 'U': column k in the upper triangular part of A contains all zeros. If UPLO = 'L': column k in the lower triangular part of A contains all zeros.

Therefore D(k,k) is exactly zero, and superdiagonal elements of column k of U (or subdiagonal elements of column k of L) are all zeros. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

NOTE: INFO only stores the first occurrence of a singularity, any subsequent occurrence of singularity is not stored in INFO even though the factorization always completes.

Related Information

For this routine in other precisions, please see *csysv_rk*, *dsysv_rk* and *zsysv_rk*. It also exists with a native C interface as *LAPACKE_ssysv_rk*.

4.6.129 ssysv_rook

SSYSV_ROOK computes the solution to a real system of linear equations

```
A * X = B,
```

where A is an N-by-N symmetric matrix and X and B are N-by-NRHS matrices.

The diagonal pivoting method is used to factor A as

```
A = U * D * U**T,  if UPLO = 'U', or
A = L * D * L**T,  if UPLO = 'L',
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

SSYTRF_ROOK is called to compute the factorization of a real symmetric matrix A using the bounded Bunch-Kaufman ("rook") diagonal pivoting method.

The factored form of A is then used to solve the system of equations A * X = B by calling SSYTRS_ROOK.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssysv_rook(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ssysv_rook_(const char *uplo, const armpl_int_t *n,
                 const armpl_int_t *nrhs, float *a, const armpl_int_t *lda,
                 armpl_int_t *ipiv, float *b, const armpl_int_t *ldb,
                 float *work, const armpl_int_t *lwork, armpl_int_t *info,
                 ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by SSYTRF_ROOK.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D, as determined by SSYTRF_ROOK.

If UPLO = 'U': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k-1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k-1 and -IPIV(k-1) were inerchaged, D(k-1:k,k-1:k) is a 2-by-2 diagonal block.

If UPLO = 'L': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k+1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k+1 and -IPIV(k+1) were inerchaged, D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= 1, and for best performance LWORK >= max(1,N*NB), where NB is the optimal blocksize for SSYTRF_ROOK.

TRS will be done with Level 2 BLAS

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, so the solution could not be computed.

Related Information

For this routine in other precisions, please see *csysv_rook*, *dsysv_rook* and *zsysv_rook*. It also exists with a native C interface as *LAPACKE_ssysv_rook*.

4.6.130 ssysvx

ssysvx uses the diagonal pivoting factorization to compute the solution to a real system of linear equations A * X = B, where A is an N-by-N symmetric matrix and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed:
```

1. If FACT = 'N', the diagonal pivoting method is used to factor A.

```
 The form of the factorization is
   A = U * D * U**T,  if UPLO = 'U', or
   A = L * D * L**T,  if UPLO = 'L',
where U (or L) is a product of permutation and unit upper (lower)
triangular matrices, and D is symmetric and block diagonal with
1-by-1 and 2-by-2 diagonal blocks.
```

2. If some D(i,i)=0, so that D is exactly singular, then the routine

```
 returns with INFO = i. Otherwise, the factored form of A is used
to estimate the condition number of the matrix A.  If the
reciprocal of the condition number is less than machine precision,
INFO = N+1 is returned as a warning, but the routine still goes on
to solve for X and compute error bounds as described below.
```

3. The system of equations is solved for X using the factored form

```
of A.
```

4. Iterative refinement is applied to improve the computed solution

```
 matrix and calculate error bounds and backward error estimates
for it.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine ssysvx(FACT, UPLO, N, NRHS, A, LDA, AF, LDAF, IPIV, B, LDB, X, LDX,
                  RCOND, FERR, BERR, WORK, LWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ssysvx_(const char *fact, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *nrhs, const float *a, const armpl_int_t *lda,
             float *af, const armpl_int_t *ldaf, armpl_int_t *ipiv,
             const float *b, const armpl_int_t *ldb, float *x,
             const armpl_int_t *ldx, float *rcond, float *ferr, float *berr,
             float *work, const armpl_int_t *lwork, armpl_int_t *iwork,
             armpl_int_t *info, ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of A has been supplied on entry. = 'F': On entry, AF and IPIV contain the factored form of A. AF and IPIV will not be modified. = 'N': The matrix A will be copied to AF and factored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). The symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input and output parameter.

AF is REAL

AF is an array, dimension (LDAF, N). If FACT = 'F', then AF is an input argument and on entry contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by SSYTRF.

If FACT = 'N', then AF is an output argument and on exit returns the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T .

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains details of the interchanges and the block structure of D, as determined by SSYTRF. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

If FACT = 'N', then IPIV is an output argument and on exit contains details of the interchanges and the block structure of D, as determined by SSYTRF.

B Input parameter.

B is REAL

B is an array, dimension (LDB, NRHS). The N-by-NRHS right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is REAL

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

The estimate of the reciprocal condition number of the matrix A. If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= max(1,3*N), and for best performance, when FACT = 'N', LWORK >= max(1,3*N,N*NB), where NB is the optimal blocksize for SSYTRF.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: D(i,i) is exactly zero. The factorization has been completed but the factor D is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+1: D is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *csysvx*, *dsysvx* and *zsysvx*. It also exists with a native C interface as *LAPACKE_ssysvx*.

4.6.131 ssysvxx

```
SSYSVXX uses the diagonal pivoting factorization to compute the
   solution to a real system of linear equations A * X = B, where A
   is an N-by-N symmetric matrix and X and B are N-by-NRHS matrices.

   If requested, both normwise and maximum componentwise error bounds
   are returned. SSYSVXX will return a solution with a tiny
   guaranteed error (O(eps) where eps is the working machine
   precision) unless the matrix is very ill-conditioned, in which
   case a warning is returned. Relevant condition numbers also are
   calculated and returned.

   SSYSVXX accepts user-provided factorizations and equilibration
   factors; see the definitions of the FACT and EQUED options.
   Solving with refinement and using a factorization from a previous
   SSYSVXX call will also produce a solution with either O(eps)
   errors or warnings, but we cannot make that claim for general
   user-provided factorizations and equilibration factors if they
   differ from what SSYSVXX would itself produce.

                                                                  The␣
↪following steps are performed:

   1. If FACT = 'E', real scaling factors are computed to equilibrate
   the system:

     diag(S)*A*diag(S)      *inv(diag(S))*X = diag(S)*B

   Whether or not the system will be equilibrated depends on the
   scaling of the matrix A, but if equilibration is used, A is
   overwritten by diag(S)*A*diag(S) and B by diag(S)*B.

   2. If FACT = 'N' or 'E', the LU decomposition is used to factor
   the matrix A (after equilibration if FACT = 'E') as

      A = U * D * U**T,  if UPLO = 'U', or
      A = L * D * L**T,  if UPLO = 'L',

   where U (or L) is a product of permutation and unit upper (lower)
   triangular matrices, and D is symmetric and block diagonal with
   1-by-1 and 2-by-2 diagonal blocks.

   3. If some D(i,i)=0, so that D is exactly singular, then the
   routine returns with INFO = i. Otherwise, the factored form of A
   is used to estimate the condition number of the matrix A (see
   argument RCOND).  If the reciprocal of the condition number is
   less than machine precision, the routine still goes on to solve
   for X and compute error bounds as described below.

   4. The system of equations is solved for X using the factored form
   of A.

   5. By default (unless PARAMS(LA_LINRX_ITREF_I) is set to zero),
   the routine will use iterative refinement to try to get a small
   error and error bounds.  Refinement calculates the residual to at
```

(continues on next page)

(continued from previous page)

```
   least twice the working precision.

   6. If equilibration was used, the matrix X is premultiplied by
   diag(R) so that it solves the original system before
   equilibration.                          Some optional parameters are bundled␣
↪in the PARAMS array.  These
    settings determine how refinement is performed, but often the
    defaults are acceptable.  If the defaults are acceptable, users
    can pass NPARAMS = 0 which prevents the source code from accessing
    the PARAMS argument.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine ssysvxx(FACT, UPLO, N, NRHS, A, LDA, AF, LDAF, IPIV, EQUED, S, B,
                   LDB, X, LDX, RCOND, RPVGRW, BERR, N_ERR_BNDS,
                   ERR_BNDS_NORM, ERR_BNDS_COMP, NPARAMS, PARAMS, WORK, IWORK,
                   INFO)
```

C specification:

```
#include "armpl.h"

void ssysvxx_(const char *fact, char *uplo, const armpl_int_t *n,
              const armpl_int_t *nrhs, float *a, const armpl_int_t *lda,
              float *af, const armpl_int_t *ldaf, armpl_int_t *ipiv,
              char *equed, float *s, float *b, const armpl_int_t *ldb,
              float *x, const armpl_int_t *ldx, float *rcond, float *rpvgrw,
              float *berr, const armpl_int_t *n_err_bnds,
              float *err_bnds_norm, float *err_bnds_comp,
              const armpl_int_t *nparams, float *params, float *work,
              armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AF and IPIV contain the factored form of A. If EQUED is not 'N', the matrix A has been equilibrated with scaling factors given by S. A, AF, and IPIV are not modified. = 'N': The matrix A will be copied to AF and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AF and factored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). The symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if FACT = 'E' and EQUED = 'Y', A is overwritten by diag(S)*A*diag(S).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input and output parameter.

AF is REAL

AF is an array, dimension (LDAF, N). If FACT = 'F', then AF is an input argument and on entry contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by SSYTRF.

If FACT = 'N', then AF is an output argument and on exit returns the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T .

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains details of the interchanges and the block structure of D, as determined by SSYTRF. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

If FACT = 'N', then IPIV is an output argument and on exit contains details of the interchanges and the block structure of D, as determined by SSYTRF.

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'Y': Both row and column equilibration, i.e., A has been replaced by diag(S) * A * diag(S). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

S Input and output parameter.

S is REAL

S is an array, dimension (N). The scale factors for A. If EQUED = 'Y', A is multiplied on the left and right by diag(S). S is an input argument if FACT = 'F'; otherwise, S is an output argument. If FACT = 'F' and EQUED = 'Y', each element of S must be positive. If S is output, each element of S is a power of the radix. If S is input, each element of S should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or

overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if EQUED = 'N', B is not modified; if EQUED = 'Y', B is overwritten by diag(S)*B;

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is REAL

X is an array, dimension (LDX, NRHS). If INFO = 0, the N-by-NRHS solution matrix X to the original system of equations. Note that A and B are modified on exit if EQUED .ne. 'N', and the solution to the equilibrated system is inv(diag(S))*X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

RPVGRW Output parameter.

RPVGRW is REAL

Reciprocal pivot growth. On exit, this contains the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If this is much less than 1, then the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, estimated condition numbers, and error bounds could be unreliable. If factorization fails with 0<INFO<=N, then this contains the reciprocal pivot growth factor for the leading INFO columns of A.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). Componentwise relative backward error. This is the componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

N_ERR_BNDS Input parameter.

N_ERR_BNDS is INTEGER

Number of error bounds to return for each right hand side and each type (normwise or componentwise). See ERR_BNDS_NORM and ERR_BNDS_COMP below.

ERR_BNDS_NORM Output parameter.

ERR_BNDS_NORM is REAL

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Output parameter.

ERR_BNDS_COMP is REAL

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:, N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

NPARAMS Input parameter.

NPARAMS is INTEGER

Specifies the number of parameters set in PARAMS. If .LE. 0, the PARAMS array is never referenced and default values are used.

PARAMS Input and output parameter.

PARAMS is REAL

PARAMS is an array, dimension NPARAMS. Specifies algorithm parameters. If an entry is .LT. 0.0, then that entry will be filled with default value used for that parameter. Only positions up to NPARAMS are accessed; defaults are used for higher-numbered parameters.

PARAMS(LA_LINRX_ITREF_I = 1) : Whether to perform iterative refinement or not. Default: 1.0 = 0.0 : No refinement is performed, and no error bounds are computed. = 1.0 : Use the double-precision refinement algorithm, possibly with doubled-single computations if the compilation environment does not support DOUBLE PRECISION. (other values are reserved for future use)

PARAMS(LA_LINRX_ITHRESH_I = 2) : Maximum number of residual computations allowed for refinement. Default: 10 Aggressive: Set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in err_bnds_norm and err_bnds_comp may no longer be trustworthy.

PARAMS(LA_LINRX_CWISE_I = 3) : Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Default: 1.0 (attempt componentwise convergence)

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (4*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. The solution to every right-hand side is guaranteed. < 0: If INFO = -i, the i-th argument had an illegal value > 0 and <= N: U(INFO, INFO) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+J: The solution corresponding to the Jth right-hand side is not guaranteed. The solutions corresponding to other right- hand sides K with K > J may not be guaranteed as well, but only the first such right-hand side is reported. If a small componentwise error is not requested (PARAMS(3) = 0.0) then the Jth right-hand side is the first with a normwise error bound that is not guaranteed (the smallest J such that ERR_BNDS_NORM(J,1) = 0.0). By default (PARAMS(3) = 1.0) the Jth right-hand side is the first with either a normwise or componentwise error bound that is not guaranteed (the smallest J such that either ERR_BNDS_NORM(J,1) = 0.0 or ERR_BNDS_COMP(J,1) = 0.0). See the definition of ERR_BNDS_NORM(:,1) and ERR_BNDS_COMP(:,1). To get information about all of the right-hand sides check ERR_BNDS_NORM or ERR_BNDS_COMP.

Related Information

For this routine in other precisions, please see *csysvxx*, *dsysvxx* and *zsysvxx*. It also exists with a native C interface as *LAPACKE_ssysvxx*.

4.6.132 ssytrs

ssytrs solves a system of linear equations A*X = B with a real symmetric matrix A using the factorization A = $U*D*U^T$ or A = $L*D*L^T$ computed by SSYTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssytrs(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void ssytrs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const float *a, const armpl_int_t *lda, const armpl_int_t *ipiv,
             float *b, const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by SSYTRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by SSYTRF.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csytrs*, *dsytrs* and *zsytrs*. It also exists with a native C interface as *LAPACKE_ssytrs*.

4.6.133 ssytrs2

`ssytrs2` solves a system of linear equations A*X = B with a real symmetric matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by SSYTRF and converted by SSYCONV.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssytrs2(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void ssytrs2_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
              const float *a, const armpl_int_t *lda, const armpl_int_t *ipiv,
              float *b, const armpl_int_t *ldb, float *work,
              armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by SSYTRF. Note that A is input / output. This might be counter-intuitive, and one may think that A is input only. A is input / output. This is because, at the start of the subroutine, we permute A in a "better" form and then we permute A back to its original form at the end.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by SSYTRF.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csytrs2*, *dsytrs2* and *zsytrs2*. It also exists with a native C interface as *LAPACKE_ssytrs2*.

4.6.134 ssytrs_3

SSYTRS_3 solves a system of linear equations A * X = B with a real symmetric matrix A using the factorization computed by SSYTRF_RK or SSYTRF_BK:

```
A = P*U*D*(U**T)*(P**T) or A = P*L*D*(L**T)*(P**T),
```

where U (or L) is unit upper (or lower) triangular matrix, U^T (or L^T) is the transpose of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This algorithm is using Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssytrs_3(UPLO, N, NRHS, A, LDA, E, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void ssytrs_3_(const char *uplo, const armpl_int_t *n,
               const armpl_int_t *nrhs, const float *a,
               const armpl_int_t *lda, const float *e,
               const armpl_int_t *ipiv, float *b, const armpl_int_t *ldb,
               armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix: = 'U': Upper triangular, form is A = P*U*D*(U^T)*(P^T); = 'L': Lower triangular, form is A = P*L*D*(L^T)*(P^T).

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). Diagonal of the block diagonal matrix D and factors U or L as computed by SSYTRF_RK and SSYTRF_BK: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D should be provided on entry in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Input parameter.

E is REAL

E is an array, dimension (N). On entry, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i),i=2:N, E(1) not referenced; If UPLO = 'L': E(i) = D(i+1,i),i=1:N-1, E(N) not referenced.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is not referenced in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by SSYTRF_RK or SSYTRF_BK.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csytrs_3*, *dsytrs_3* and *zsytrs_3*. It also exists with a native C interface as *LAPACKE_ssytrs_3*.

4.6.135 ssytrs_aa

SSYTRS_AA solves a system of linear equations A*X = B with a real symmetric matrix A using the factorization A = U*T*U^T or A = L*T*L^T computed by SSYTRF_AA.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssytrs_aa(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ssytrs_aa_(const char *uplo, const armpl_int_t *n,
                const armpl_int_t *nrhs, const float *a,
                const armpl_int_t *lda, const armpl_int_t *ipiv, float *b,
                const armpl_int_t *ldb, const float *work,
                const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*T*U^T ; = 'L': Lower triangular, form is A = L*T*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). Details of factors computed by SSYTRF_AA.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges as computed by SSYTRF_AA.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Input parameter.

WORK is DOUBLE array, dimension (MAX(1, LWORK))

LWORK Input parameter.

LWORK is INTEGER, LWORK >= MAX(1,3*N-2).

param[out] INFO verbatim INFO is INTEGER = 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csytrs_aa*, *dsytrs_aa* and *zsytrs_aa*. It also exists with a native C interface as *LAPACKE_ssytrs_aa*.

4.6.136 ssytrs_rook

SSYTRS_ROOK solves a system of linear equations A*X = B with a real symmetric matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by SSYTRF_ROOK.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssytrs_rook(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void ssytrs_rook_(const char *uplo, const armpl_int_t *n,
                  const armpl_int_t *nrhs, const float *a,
                  const armpl_int_t *lda, const armpl_int_t *ipiv, float *b,
                  const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by SSYTRF_ROOK.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by SSYTRF_ROOK.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csytrs_rook*, *dsytrs_rook* and *zsytrs_rook*. It also exists with a native C interface as *LAPACKE_ssytrs_rook*.

4.6.137 stbtrs

stbtrs solves a triangular system of the form

```
A * X = B  or  A**T * X = B,
```

where A is a triangular band matrix of order N, and B is an N-by NRHS matrix. A check is made to verify that A is nonsingular.

Syntax

Fortran specification:

```
use armpl_library

subroutine stbtrs(UPLO, TRANS, DIAG, N, KD, NRHS, AB, LDAB, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void stbtrs_(const char *uplo, const char *trans, const char *diag,
             const armpl_int_t *n, const armpl_int_t *kd,
             const armpl_int_t *nrhs, const float *ab,
             const armpl_int_t *ldab, float *b, const armpl_int_t *ldb,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose = Transpose)

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals or subdiagonals of the triangular band matrix A. KD >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AB Input parameter.

AB is REAL

AB is an array, dimension (LDAB, N). The upper or lower triangular band matrix A, stored in the first kd+1 rows of AB. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd). If DIAG = 'U', the diagonal elements of A are not referenced and are assumed to be 1.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, if INFO = 0, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element of A is zero, indicating that the matrix is singular and the solutions X have not been computed.

Related Information

For this routine in other precisions, please see *ctbtrs*, *dtbtrs* and *ztbtrs*. It also exists with a native C interface as *LAPACKE_stbtrs*.

4.6.138 stptrs

stptrs solves a triangular system of the form

```
A * X = B  or  A**T * X = B,
```

where A is a triangular matrix of order N stored in packed format, and B is an N-by-NRHS matrix. A check is made to verify that A is nonsingular.

Syntax

Fortran specification:

```
use armpl_library

subroutine stptrs(UPLO, TRANS, DIAG, N, NRHS, AP, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void stptrs_(const char *uplo, const char *trans, const char *diag,
             const armpl_int_t *n, const armpl_int_t *nrhs, const float *ap,
             float *b, const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose = Transpose)

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AP Input parameter.

AP is REAL

AP is an array, dimension (N*(N+1)/2). The upper or lower triangular matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, if INFO = 0, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element of A is zero, indicating that the matrix is singular and the solutions X have not been computed.

Related Information

For this routine in other precisions, please see *ctptrs*, *dtptrs* and *ztptrs*. It also exists with a native C interface as *LAPACKE_stptrs*.

4.6.139 zcgesv

zcgesv computes the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N matrix and X and B are N-by-NRHS matrices.

zcgesv first attempts to factorize the matrix in COMPLEX and use this factorization within an iterative refinement procedure to produce a solution with COMPLEX*16 normwise backward error quality (see below). If the approach fails the method switches to a COMPLEX*16 factorization and solve.

The iterative refinement is not going to be a winning strategy if the ratio COMPLEX performance over COMPLEX*16 performance is too small. A reasonable strategy should take the number of right-hand sides and the size of the matrix into account. This might be done with a call to ILAENV in the future. Up to now, we always try iterative refinement.

The iterative refinement process is stopped if

```
ITER > ITERMAX
```

or for all the RHS we have:

```
RNRM < SQRT(N)*XNRM*ANRM*EPS*BWDMAX
```

where

```
 o ITER is the number of the current iteration in the iterative
  refinement process
o RNRM is the infinity-norm of the residual
o XNRM is the infinity-norm of the solution
o ANRM is the infinity-operator-norm of the matrix A
o EPS is the machine epsilon returned by DLAMCH('Epsilon')
```

The value ITERMAX and BWDMAX are fixed to 30 and 1.0D+00 respectively.

Syntax

Fortran specification:

```
use armpl_library

subroutine zcgesv(N, NRHS, A, LDA, IPIV, B, LDB, X, LDX, WORK, SWORK, RWORK,
                  ITER, INFO)
```

C specification:

```
#include "armpl.h"

void zcgesv_(const armpl_int_t *n, const armpl_int_t *nrhs,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_int_t *ipiv, const armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, armpl_doublecomplex_t *x,
             const armpl_int_t *ldx, armpl_doublecomplex_t *work,
             armpl_singlecomplex_t *swork, double *rwork, armpl_int_t *iter,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array,. dimension (LDA, N) On entry, the N-by-N coefficient matrix A. On exit, if iterative refinement has been successfully used (INFO.EQ.0 and ITER.GE.0, see description below), then A is unchanged, if double precision factorization has been used (INFO.EQ.0 and ITER.LT.0, see description below), then the array A contains the factors L and U from the factorization A = P*L*U; the unit diagonal elements of L are not stored.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices that define the permutation matrix P; row i of the matrix was interchanged with row IPIV(i). Corresponds either to the single precision factorization (if INFO.EQ.0 and ITER.GE.0) or the double precision factorization (if INFO.EQ.0 and ITER.LT.0).

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). The N-by-NRHS right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). If INFO = 0, the N-by-NRHS solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N, NRHS). This array is used to hold the residual vectors.

SWORK Output parameter.

SWORK is COMPLEX

SWORK is an array, dimension (N*(N+NRHS)). This array is used to use the single precision matrix and the right-hand sides or solutions in single precision.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

ITER Output parameter.

ITER is INTEGER

< 0: iterative refinement has failed, COMPLEX*16 factorization has been performed -1 : the routine fell back to full precision for implementation- or machine-specific reasons -2 : narrowing the precision induced an overflow, the routine fell back to full precision -3 : failure of CGETRF -31: stop the iterative refinement after the 30th iterations > 0: iterative refinement has been successfully used. Returns the number of iterations

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, U(i,i) computed in COMPLEX*16 is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

Related Information

It also exists with a native C interface as *LAPACKE_zcgesv*.

4.6.140 zcposv

zcposv computes the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N Hermitian positive definite matrix and X and B are N-by-NRHS matrices.

zcposv first attempts to factorize the matrix in COMPLEX and use this factorization within an iterative refinement procedure to produce a solution with COMPLEX*16 normwise backward error quality (see below). If the approach fails the method switches to a COMPLEX*16 factorization and solve.

The iterative refinement is not going to be a winning strategy if the ratio COMPLEX performance over COMPLEX*16 performance is too small. A reasonable strategy should take the number of right-hand sides and the size of the matrix into account. This might be done with a call to ILAENV in the future. Up to now, we always try iterative refinement.

The iterative refinement process is stopped if

```
ITER > ITERMAX
```

or for all the RHS we have:

```
RNRM < SQRT(N)*XNRM*ANRM*EPS*BWDMAX
```

where

```
 o ITER is the number of the current iteration in the iterative
  refinement process
o RNRM is the infinity-norm of the residual
o XNRM is the infinity-norm of the solution
o ANRM is the infinity-operator-norm of the matrix A
o EPS is the machine epsilon returned by DLAMCH('Epsilon')
```

The value ITERMAX and BWDMAX are fixed to 30 and 1.0D+00 respectively.

Syntax

Fortran specification:

```
use armpl_library

subroutine zcposv(UPLO, N, NRHS, A, LDA, B, LDB, X, LDX, WORK, SWORK, RWORK,
                  ITER, INFO)
```

C specification:

```
#include "armpl.h"

void zcposv_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_doublecomplex_t *b, const armpl_int_t *ldb,
             armpl_doublecomplex_t *x, const armpl_int_t *ldx,
             armpl_doublecomplex_t *work, armpl_singlecomplex_t *swork,
             double *rwork, armpl_int_t *iter, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array,. dimension (LDA, N) On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

Note that the imaginary parts of the diagonal elements need not be set and are assumed to be zero.

On exit, if iterative refinement has been successfully used (INFO.EQ.0 and ITER.GE.0, see description below), then A is unchanged, if double precision factorization has been used (INFO.EQ.0 and ITER.LT.0, see description below), then the array A contains the factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H .

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). The N-by-NRHS right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). If INFO = 0, the N-by-NRHS solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N, NRHS). This array is used to hold the residual vectors.

SWORK Output parameter.

SWORK is COMPLEX

SWORK is an array, dimension (N*(N+NRHS)). This array is used to use the single precision matrix and the right-hand sides or solutions in single precision.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

ITER Output parameter.

ITER is INTEGER

< 0: iterative refinement has failed, COMPLEX*16 factorization has been performed -1 : the routine fell back to full precision for implementation- or machine-specific reasons -2 : narrowing the precision induced an overflow, the routine fell back to full precision -3 : failure of CPOTRF -31: stop the iterative refinement after the 30th iterations > 0: iterative refinement has been successfully used. Returns the number of iterations

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i of (COMPLEX*16) A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

Related Information

It also exists with a native C interface as *LAPACKE_zcposv*.

4.6.141 zgbsv

zgbsv computes the solution to a complex system of linear equations A * X = B, where A is a band matrix of order N with KL subdiagonals and KU superdiagonals, and X and B are N-by-NRHS matrices.

The LU decomposition with partial pivoting and row interchanges is used to factor A as A = L * U, where L is a product of permutation and unit lower triangular matrices with KL subdiagonals, and U is upper triangular with KL+KU superdiagonals. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgbsv(N, KL, KU, NRHS, AB, LDAB, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void zgbsv_(const armpl_int_t *n, const armpl_int_t *kl,
            const armpl_int_t *ku, const armpl_int_t *nrhs,
            armpl_doublecomplex_t *ab, const armpl_int_t *ldab,
            armpl_int_t *ipiv, armpl_doublecomplex_t *b,
            const armpl_int_t *ldb, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AB Input and output parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows KL+1 to 2*KL+KU+1; rows 1 to KL of the array need not be set. The j-th column of A is stored in the j-th column of the array AB as follows: AB(KL+KU+1+i-j,j) = A(i,j) for max(1,j-KU)<=i<=min(N,j+KL) On exit, details of the factorization: U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1. See below for further details.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= 2*KL+KU+1.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices that define the permutation matrix P; row i of the matrix was interchanged with row IPIV(i).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and the solution has not been computed.

Related Information

For this routine in other precisions, please see *cgbsv*, *dgbsv* and *sgbsv*. It also exists with a native C interface as *LAPACKE_zgbsv*.

4.6.142 zgbsvx

zgbsvx uses the LU factorization to compute the solution to a complex system of linear equations A * X = B, A^T * X = B, or A^H * X = B, where A is a band matrix of order N with KL subdiagonals and KU superdiagonals, and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed by this subroutine:
```

1. If FACT = 'E', real scaling factors are computed to equilibrate

```
 the system:
   TRANS = 'N':  diag(R)*A*diag(C)      *inv(diag(C))*X = diag(R)*B
   TRANS = 'T': (diag(R)*A*diag(C))**T *inv(diag(R))*X = diag(C)*B
   TRANS = 'C': (diag(R)*A*diag(C))**H *inv(diag(R))*X = diag(C)*B
Whether or not the system will be equilibrated depends on the
scaling of the matrix A, but if equilibration is used, A is
overwritten by diag(R)*A*diag(C) and B by diag(R)*B (if TRANS='N')
or diag(C)*B (if TRANS = 'T' or 'C').
```

2. If FACT = 'N' or 'E', the LU decomposition is used to factor the

```
 matrix A (after equilibration if FACT = 'E') as
   A = L * U,
where L is a product of permutation and unit lower triangular
matrices with KL subdiagonals, and U is upper triangular with
KL+KU superdiagonals.
```

3. If some U(i,i)=0, so that U is exactly singular, then the routine

```
 returns with INFO = i. Otherwise, the factored form of A is used
to estimate the condition number of the matrix A.  If the
reciprocal of the condition number is less than machine precision,
INFO = N+1 is returned as a warning, but the routine still goes on
to solve for X and compute error bounds as described below.
```

4. The system of equations is solved for X using the factored form

```
of A.
```

5. Iterative refinement is applied to improve the computed solution

```
 matrix and calculate error bounds and backward error estimates
for it.
```

6. If equilibration was used, the matrix X is premultiplied by

```
 diag(C) (if TRANS = 'N') or diag(R) (if TRANS = 'T' or 'C') so
that it solves the original system before equilibration.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zgbsvx(FACT, TRANS, N, KL, KU, NRHS, AB, LDAB, AFB, LDAFB, IPIV,
                  EQUED, R, C, B, LDB, X, LDX, RCOND, FERR, BERR, WORK, RWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void zgbsvx_(const char *fact, const char *trans, const armpl_int_t *n,
             const armpl_int_t *kl, const armpl_int_t *ku,
             const armpl_int_t *nrhs, armpl_doublecomplex_t *ab,
             const armpl_int_t *ldab, armpl_doublecomplex_t *afb,
             const armpl_int_t *ldafb, armpl_int_t *ipiv, char *equed,
             double *r, double *c, armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, armpl_doublecomplex_t *x,
             const armpl_int_t *ldx, double *rcond, double *ferr,
             double *berr, armpl_doublecomplex_t *work, double *rwork,
             armpl_int_t *info, ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AFB and IPIV contain the factored form of A. If EQUED is not 'N', the matrix A has been equilibrated with scaling factors given by R and C. AB, AFB, and IPIV are not modified. = 'N': The matrix A will be copied to AFB and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AFB and factored.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations. = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose)

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AB Input and output parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(KU+1+i-j,j) = A(i,j) for max(1,j-KU)<=i<=min(N,j+kl)

If FACT = 'F' and EQUED is not 'N', then A must have been equilibrated by the scaling factors in R and/or C. AB is not modified if FACT = 'F' or 'N', or if FACT = 'E' and EQUED = 'N' on exit.

On exit, if EQUED .ne. 'N', A is scaled as follows: EQUED = 'R': A := diag(R) * A EQUED = 'C': A := A * diag(C) EQUED = 'B': A := diag(R) * A * diag(C).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KL+KU+1.

AFB Input and output parameter.

AFB is COMPLEX*16

AFB is an array, dimension (LDAFB, N). If FACT = 'F', then AFB is an input argument and on entry contains details of the LU factorization of the band matrix A, as computed by ZGBTRF. U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1. If EQUED .ne. 'N', then AFB is the factored form of the equilibrated matrix A.

If FACT = 'N', then AFB is an output argument and on exit returns details of the LU factorization of A.

If FACT = 'E', then AFB is an output argument and on exit returns details of the LU factorization of the equilibrated matrix A (see the description of AB for the form of the equilibrated matrix).

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AFB. LDAFB >= 2*KL+KU+1.

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains the pivot indices from the factorization A = L*U as computed by ZGBTRF; row i of the matrix was interchanged with row IPIV(i).

If FACT = 'N', then IPIV is an output argument and on exit contains the pivot indices from the factorization A = L*U of the original matrix A.

If FACT = 'E', then IPIV is an output argument and on exit contains the pivot indices from the factorization A = L*U of the equilibrated matrix A.

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'R': Row equilibration, i.e., A has been premultiplied by diag(R). = 'C': Column equilibration, i.e., A has been postmultiplied by diag(C). = 'B': Both row and column equilibration, i.e., A has been replaced by diag(R) * A * diag(C). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

R Input and output parameter.

R is DOUBLE PRECISION

R is an array, dimension (N). The row scale factors for A. If EQUED = 'R' or 'B', A is multiplied on the left by diag(R); if EQUED = 'N' or 'C', R is not accessed. R is an input argument if FACT = 'F'; otherwise, R is an output argument. If FACT = 'F' and EQUED = 'R' or 'B', each element of R must be positive.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). The column scale factors for A. If EQUED = 'C' or 'B', A is multiplied on the right by diag(C); if EQUED = 'N' or 'R', C is not accessed. C is an input argument if FACT = 'F'; otherwise, C is an output argument. If FACT = 'F' and EQUED = 'C' or 'B', each element of C must be positive.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, if EQUED = 'N', B is not modified; if TRANS = 'N' and EQUED = 'R' or 'B', B is overwritten by diag(R)*B; if TRANS = 'T' or 'C' and EQUED = 'C' or 'B', B is overwritten by diag(C)*B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X to the original system of equations. Note that A and B are modified on exit if EQUED .ne. 'N', and the solution to the equilibrated system is inv(diag(C))* X if TRANS = 'N' and EQUED = 'C' or 'B', or inv(diag(R))* X if TRANS = 'T' or 'C' and EQUED = 'R' or 'B'.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The estimate of the reciprocal condition number of the matrix A after equilibration (if done). If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N). On exit, RWORK(1) contains the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If RWORK(1) is much less than 1, then the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, condition estimator RCOND, and forward error bound FERR could be unreliable. If factorization fails with 0<INFO<=N, then RWORK(1) contains the reciprocal pivot growth factor for the leading INFO columns of A.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+1: U is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *cgbsvx*, *dgbsvx* and *sgbsvx*. It also exists with a native C interface as *LAPACKE_zgbsvx*.

4.6.143 zgbsvxx

```
ZGBSVXX uses the LU factorization to compute the solution to a
   complex*16 system of linear equations  A * X = B,  where A is an
   N-by-N matrix and X and B are N-by-NRHS matrices.

   If requested, both normwise and maximum componentwise error bounds
   are returned. ZGBSVXX will return a solution with a tiny
   guaranteed error (O(eps) where eps is the working machine
   precision) unless the matrix is very ill-conditioned, in which
   case a warning is returned. Relevant condition numbers also are
   calculated and returned.

   ZGBSVXX accepts user-provided factorizations and equilibration
   factors; see the definitions of the FACT and EQUED options.
   Solving with refinement and using a factorization from a previous
   ZGBSVXX call will also produce a solution with either O(eps)
   errors or warnings, but we cannot make that claim for general
   user-provided factorizations and equilibration factors if they
   differ from what ZGBSVXX would itself produce.

                                                                  The␣
↪following steps are performed:

   1. If FACT = 'E', double precision scaling factors are computed to equilibrate
   the system:

     TRANS = 'N':  diag(R)*A*diag(C)     *inv(diag(C))*X = diag(R)*B
     TRANS = 'T': (diag(R)*A*diag(C))**T *inv(diag(R))*X = diag(C)*B
     TRANS = 'C': (diag(R)*A*diag(C))**H *inv(diag(R))*X = diag(C)*B

   Whether or not the system will be equilibrated depends on the
   scaling of the matrix A, but if equilibration is used, A is
   overwritten by diag(R)*A*diag(C) and B by diag(R)*B (if TRANS='N')
```

(continues on next page)

(continued from previous page)

```
   or diag(C)*B (if TRANS = 'T' or 'C').

   2. If FACT = 'N' or 'E', the LU decomposition is used to factor
   the matrix A (after equilibration if FACT = 'E') as

     A = P * L * U,

   where P is a permutation matrix, L is a unit lower triangular
   matrix, and U is upper triangular.

   3. If some U(i,i)=0, so that U is exactly singular, then the
   routine returns with INFO = i. Otherwise, the factored form of A
   is used to estimate the condition number of the matrix A (see
   argument RCOND). If the reciprocal of the condition number is less
   than machine precision, the routine still goes on to solve for X
   and compute error bounds as described below.

   4. The system of equations is solved for X using the factored form
   of A.

   5. By default (unless PARAMS(LA_LINRX_ITREF_I) is set to zero),
   the routine will use iterative refinement to try to get a small
   error and error bounds.  Refinement calculates the residual to at
   least twice the working precision.

   6. If equilibration was used, the matrix X is premultiplied by
   diag(C) (if TRANS = 'N') or diag(R) (if TRANS = 'T' or 'C') so
   that it solves the original system before equilibration.
↪    Some optional parameters are bundled in the PARAMS array.  These
    settings determine how refinement is performed, but often the
    defaults are acceptable.  If the defaults are acceptable, users
    can pass NPARAMS = 0 which prevents the source code from accessing
    the PARAMS argument.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zgbsvxx(FACT, TRANS, N, KL, KU, NRHS, AB, LDAB, AFB, LDAFB, IPIV,
                   EQUED, R, C, B, LDB, X, LDX, RCOND, RPVGRW, BERR,
                   N_ERR_BNDS, ERR_BNDS_NORM, ERR_BNDS_COMP, NPARAMS, PARAMS,
                   WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgbsvxx_(const char *fact, const char *trans, const armpl_int_t *n,
              const armpl_int_t *kl, const armpl_int_t *ku,
              const armpl_int_t *nrhs, armpl_doublecomplex_t *ab,
              const armpl_int_t *ldab, armpl_doublecomplex_t *afb,
              const armpl_int_t *ldafb, armpl_int_t *ipiv, char *equed,
              double *r, double *c, armpl_doublecomplex_t *b,
              const armpl_int_t *ldb, armpl_doublecomplex_t *x,
              const armpl_int_t *ldx, double *rcond, double *rpvgrw,
              double *berr, const armpl_int_t *n_err_bnds,
              double *err_bnds_norm, double *err_bnds_comp,
              const armpl_int_t *nparams, double *params,
```

(continues on next page)

(continued from previous page)

```
                armpl_doublecomplex_t *work, double *rwork, armpl_int_t *info,
                ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AF and IPIV contain the factored form of A. If EQUED is not 'N', the matrix A has been equilibrated with scaling factors given by R and C. A, AF, and IPIV are not modified. = 'N': The matrix A will be copied to AF and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AF and factored.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate Transpose = Transpose)

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AB Input and output parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(KU+1+i-j,j) = A(i,j) for max(1,j-KU)<=i<=min(N,j+kl)

If FACT = 'F' and EQUED is not 'N', then AB must have been equilibrated by the scaling factors in R and/or C. AB is not modified if FACT = 'F' or 'N', or if FACT = 'E' and EQUED = 'N' on exit.

On exit, if EQUED .ne. 'N', A is scaled as follows: EQUED = 'R': A := diag(R) * A EQUED = 'C': A := A * diag(C) EQUED = 'B': A := diag(R) * A * diag(C).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KL+KU+1.

AFB Input and output parameter.

AFB is COMPLEX*16

AFB is an array, dimension (LDAFB, N). If FACT = 'F', then AFB is an input argument and on entry contains details of the LU factorization of the band matrix A, as computed by ZGBTRF. U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1. If EQUED .ne. 'N', then AFB is the factored form of the equilibrated matrix A.

If FACT = 'N', then AF is an output argument and on exit returns the factors L and U from the factorization A = P*L*U of the original matrix A.

If FACT = 'E', then AF is an output argument and on exit returns the factors L and U from the factorization A = P*L*U of the equilibrated matrix A (see the description of A for the form of the equilibrated matrix).

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AFB. LDAFB >= 2*KL+KU+1.

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains the pivot indices from the factorization A = P*L*U as computed by DGETRF; row i of the matrix was interchanged with row IPIV(i).

If FACT = 'N', then IPIV is an output argument and on exit contains the pivot indices from the factorization A = P*L*U of the original matrix A.

If FACT = 'E', then IPIV is an output argument and on exit contains the pivot indices from the factorization A = P*L*U of the equilibrated matrix A.

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'R': Row equilibration, i.e., A has been premultiplied by diag(R). = 'C': Column equilibration, i.e., A has been postmultiplied by diag(C). = 'B': Both row and column equilibration, i.e., A has been replaced by diag(R) * A * diag(C). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

R Input and output parameter.

R is DOUBLE PRECISION

R is an array, dimension (N). The row scale factors for A. If EQUED = 'R' or 'B', A is multiplied on the left by diag(R); if EQUED = 'N' or 'C', R is not accessed. R is an input argument if FACT = 'F'; otherwise, R is an output argument. If FACT = 'F' and EQUED = 'R' or 'B', each element of R must be positive. If R is output, each element of R is a power of the radix. If R is input, each element of R should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). The column scale factors for A. If EQUED = 'C' or 'B', A is multiplied on the right by diag(C); if EQUED = 'N' or 'R', C is not accessed. C is an input argument if FACT = 'F'; otherwise, C is an output argument. If FACT = 'F' and EQUED = 'C' or 'B', each element of C must be positive. If C is output, each element of C is a power of the radix. If C is input, each element of C should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if EQUED = 'N', B is not modified; if TRANS = 'N' and EQUED = 'R' or 'B', B is overwritten by diag(R)*B; if TRANS = 'T' or 'C' and EQUED = 'C' or 'B', B is overwritten by diag(C)*B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). If INFO = 0, the N-by-NRHS solution matrix X to the original system of equations. Note that A and B are modified on exit if EQUED .ne. 'N', and the solution to the equilibrated system is inv(diag(C))* X if TRANS = 'N' and EQUED = 'C' or 'B', or inv(diag(R))* X if TRANS = 'T' or 'C' and EQUED = 'R' or 'B'.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

RPVGRW Output parameter.

RPVGRW is DOUBLE PRECISION

Reciprocal pivot growth. On exit, this contains the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If this is much less than 1, then the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, estimated condition numbers, and error bounds could be unreliable. If factorization fails with 0<INFO<=N, then this contains the reciprocal pivot growth factor for the leading INFO columns of A. In DGESVX, this quantity is returned in WORK(1).

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). Componentwise relative backward error. This is the componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

N_ERR_BNDS Input parameter.

N_ERR_BNDS is INTEGER

Number of error bounds to return for each right hand side and each type (normwise or componentwise). See ERR_BNDS_NORM and ERR_BNDS_COMP below.

ERR_BNDS_NORM Output parameter.

ERR_BNDS_NORM is DOUBLE PRECISION

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * dlamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * dlamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * dlamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Output parameter.

ERR_BNDS_COMP is DOUBLE PRECISION

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:, N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * dlamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * dlamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * dlamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

NPARAMS Input parameter.

NPARAMS is INTEGER

Specifies the number of parameters set in PARAMS. If .LE. 0, the PARAMS array is never referenced and default values are used.

PARAMS Input and output parameter.

PARAMS is DOUBLE PRECISION

PARAMS is an array, dimension NPARAMS. Specifies algorithm parameters. If an entry is .LT. 0.0, then that entry will be filled with default value used for that parameter. Only positions up to NPARAMS are accessed; defaults are used for higher-numbered parameters.

PARAMS(LA_LINRX_ITREF_I = 1) : Whether to perform iterative refinement or not. Default: 1.0D+0 = 0.0 : No refinement is performed, and no error bounds are computed. = 1.0 : Use the extra-precise refinement algorithm. (other values are reserved for future use)

PARAMS(LA_LINRX_ITHRESH_I = 2) : Maximum number of residual computations allowed for refinement. Default: 10 Aggressive: Set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in err_bnds_norm and err_bnds_comp may no longer be trustworthy.

PARAMS(LA_LINRX_CWISE_I = 3) : Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Default: 1.0 (attempt componentwise convergence)

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. The solution to every right-hand side is guaranteed. < 0: If INFO = -i, the i-th argument had an illegal value > 0 and <= N: U(INFO, INFO) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+J: The solution corresponding to the Jth right-hand side is not guaranteed. The solutions corresponding to other right- hand sides K with K > J may not be guaranteed as well, but only the first such right-hand side is reported. If a small componentwise error is not requested (PARAMS(3) = 0.0) then the Jth right-hand side is the first with a normwise error bound that is not guaranteed (the smallest J such that ERR_BNDS_NORM(J,1) = 0.0). By default (PARAMS(3) = 1.0) the Jth right-hand side is the first with either a normwise or componentwise error bound that is not guaranteed (the smallest J such that either ERR_BNDS_NORM(J,1) = 0.0 or ERR_BNDS_COMP(J,1) = 0.0). See the definition of ERR_BNDS_NORM(:,1) and ERR_BNDS_COMP(:,1). To get information about all of the right-hand sides check ERR_BNDS_NORM or ERR_BNDS_COMP.

Related Information

For this routine in other precisions, please see *cgbsvxx*, *dgbsvxx* and *sgbsvxx*. It also exists with a native C interface as *LAPACKE_zgbsvxx*.

4.6.144 zgbtrs

zgbtrs solves a system of linear equations

```
A * X = B,  A**T * X = B,  or  A**H * X = B
```

with a general band matrix A using the LU factorization computed by ZGBTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgbtrs(TRANS, N, KL, KU, NRHS, AB, LDAB, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void zgbtrs_(const char *trans, const armpl_int_t *n, const armpl_int_t *kl,
             const armpl_int_t *ku, const armpl_int_t *nrhs,
             const armpl_doublecomplex_t *ab, const armpl_int_t *ldab,
             const armpl_int_t *ipiv, armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations. = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose)

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AB Input parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). Details of the LU factorization of the band matrix A, as computed by ZGBTRF. U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= 2*KL+KU+1.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices; for 1 <= i <= N, row i of the matrix was interchanged with row IPIV(i).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgbtrs*, *dgbtrs* and *sgbtrs*. It also exists with a native C interface as *LAPACKE_zgbtrs*.

4.6.145 zgesv

zgesv computes the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N matrix and X and B are N-by-NRHS matrices.

The LU decomposition with partial pivoting and row interchanges is used to factor A as

```
A = P * L * U,
```

where P is a permutation matrix, L is unit lower triangular, and U is upper triangular. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgesv(N, NRHS, A, LDA, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void zgesv_(const armpl_int_t *n, const armpl_int_t *nrhs,
            armpl_doublecomplex_t *a, const armpl_int_t *lda,
            armpl_int_t *ipiv, armpl_doublecomplex_t *b,
            const armpl_int_t *ldb, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the N-by-N coefficient matrix A. On exit, the factors L and U from the factorization A = P*L*U; the unit diagonal elements of L are not stored.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices that define the permutation matrix P; row i of the matrix was interchanged with row IPIV(i).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

Related Information

For this routine in other precisions, please see *cgesv*, *dgesv* and *sgesv*. It also exists with a native C interface as *LAPACKE_zgesv*.

4.6.146 zgesvx

zgesvx uses the LU factorization to compute the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N matrix and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed:
```

1. If FACT = 'E', real scaling factors are computed to equilibrate

```
 the system:
   TRANS = 'N':  diag(R)*A*diag(C)      *inv(diag(C))*X = diag(R)*B
   TRANS = 'T': (diag(R)*A*diag(C))**T *inv(diag(R))*X = diag(C)*B
   TRANS = 'C': (diag(R)*A*diag(C))**H *inv(diag(R))*X = diag(C)*B
Whether or not the system will be equilibrated depends on the
scaling of the matrix A, but if equilibration is used, A is
overwritten by diag(R)*A*diag(C) and B by diag(R)*B (if TRANS='N')
or diag(C)*B (if TRANS = 'T' or 'C').
```

2. If FACT = 'N' or 'E', the LU decomposition is used to factor the

```
 matrix A (after equilibration if FACT = 'E') as
   A = P * L * U,
where P is a permutation matrix, L is a unit lower triangular
matrix, and U is upper triangular.
```

3. If some U(i,i)=0, so that U is exactly singular, then the routine

```
 returns with INFO = i. Otherwise, the factored form of A is used
to estimate the condition number of the matrix A.  If the
reciprocal of the condition number is less than machine precision,
INFO = N+1 is returned as a warning, but the routine still goes on
to solve for X and compute error bounds as described below.
```

4. The system of equations is solved for X using the factored form

```
of A.
```

5. Iterative refinement is applied to improve the computed solution

```
 matrix and calculate error bounds and backward error estimates
for it.
```

6. If equilibration was used, the matrix X is premultiplied by

```
 diag(C) (if TRANS = 'N') or diag(R) (if TRANS = 'T' or 'C') so
that it solves the original system before equilibration.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zgesvx(FACT, TRANS, N, NRHS, A, LDA, AF, LDAF, IPIV, EQUED, R, C,
                  B, LDB, X, LDX, RCOND, FERR, BERR, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgesvx_(const char *fact, const char *trans, const armpl_int_t *n,
             const armpl_int_t *nrhs, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_doublecomplex_t *af,
             const armpl_int_t *ldaf, armpl_int_t *ipiv, char *equed,
             double *r, double *c, armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, armpl_doublecomplex_t *x,
             const armpl_int_t *ldx, double *rcond, double *ferr,
             double *berr, armpl_doublecomplex_t *work, double *rwork,
             armpl_int_t *info, ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AF and IPIV contain the factored form of A. If EQUED is not 'N', the matrix A has been equilibrated with scaling factors given by R and C. A, AF, and IPIV are not modified. = 'N': The matrix A will be copied to AF and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AF and factored.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose)

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A. If FACT = 'F' and EQUED is not 'N', then A must have been equilibrated by the scaling factors in R and/or C. A is not modified if FACT = 'F' or 'N', or if FACT = 'E' and EQUED = 'N' on exit.

On exit, if EQUED .ne. 'N', A is scaled as follows: EQUED = 'R': A := diag(R) * A EQUED = 'C': A := A * diag(C) EQUED = 'B': A := diag(R) * A * diag(C).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input and output parameter.

AF is COMPLEX*16

AF is an array, dimension (LDAF, N). If FACT = 'F', then AF is an input argument and on entry contains the factors L and U from the factorization A = P*L*U as computed by ZGETRF. If EQUED .ne. 'N', then AF is the factored form of the equilibrated matrix A.

If FACT = 'N', then AF is an output argument and on exit returns the factors L and U from the factorization A = P*L*U of the original matrix A.

If FACT = 'E', then AF is an output argument and on exit returns the factors L and U from the factorization A = P*L*U of the equilibrated matrix A (see the description of A for the form of the equilibrated matrix).

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains the pivot indices from the factorization A = P*L*U as computed by ZGETRF; row i of the matrix was interchanged with row IPIV(i).

If FACT = 'N', then IPIV is an output argument and on exit contains the pivot indices from the factorization A = P*L*U of the original matrix A.

If FACT = 'E', then IPIV is an output argument and on exit contains the pivot indices from the factorization A = P*L*U of the equilibrated matrix A.

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'R': Row equilibration, i.e., A has been premultiplied by diag(R). = 'C': Column equilibration, i.e., A has been postmultiplied by diag(C). = 'B': Both row and column equilibration, i.e., A has been replaced by diag(R) * A * diag(C). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

R Input and output parameter.

R is DOUBLE PRECISION

R is an array, dimension (N). The row scale factors for A. If EQUED = 'R' or 'B', A is multiplied on the left by diag(R); if EQUED = 'N' or 'C', R is not accessed. R is an input argument if FACT = 'F'; otherwise, R is an output argument. If FACT = 'F' and EQUED = 'R' or 'B', each element of R must be positive.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). The column scale factors for A. If EQUED = 'C' or 'B', A is multiplied on the right by diag(C); if EQUED = 'N' or 'R', C is not accessed. C is an input argument if FACT = 'F'; otherwise, C is an output argument. If FACT = 'F' and EQUED = 'C' or 'B', each element of C must be positive.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if EQUED = 'N', B is not modified; if TRANS = 'N' and EQUED = 'R' or 'B', B is overwritten by diag(R)*B; if TRANS = 'T' or 'C' and EQUED = 'C' or 'B', B is overwritten by diag(C)*B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X to the original system of equations. Note that A and B are modified on exit if EQUED .ne. 'N', and the solution to the equilibrated system is inv(diag(C))* X if TRANS = 'N' and EQUED = 'C' or 'B', or inv(diag(R))* X if TRANS = 'T' or 'C' and EQUED = 'R' or 'B'.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The estimate of the reciprocal condition number of the matrix A after equilibration (if done). If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (2*N). On exit, RWORK(1) contains the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If RWORK(1) is much less than 1, then the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, condition estimator RCOND, and forward error bound FERR could be unreliable. If factorization fails with 0<INFO<=N, then RWORK(1) contains the reciprocal pivot growth factor for the leading INFO columns of A.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+1: U is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *cgesvx*, *dgesvx* and *sgesvx*. It also exists with a native C interface as *LAPACKE_zgesvx*.

4.6.147 zgesvxx

```
ZGESVXX uses the LU factorization to compute the solution to a
   complex*16 system of linear equations  A * X = B,  where A is an
   N-by-N matrix and X and B are N-by-NRHS matrices.

   If requested, both normwise and maximum componentwise error bounds
   are returned. ZGESVXX will return a solution with a tiny
   guaranteed error (O(eps) where eps is the working machine
   precision) unless the matrix is very ill-conditioned, in which
   case a warning is returned. Relevant condition numbers also are
   calculated and returned.
```

(continues on next page)

(continued from previous page)

```
   ZGESVXX accepts user-provided factorizations and equilibration
   factors; see the definitions of the FACT and EQUED options.
   Solving with refinement and using a factorization from a previous
   ZGESVXX call will also produce a solution with either O(eps)
   errors or warnings, but we cannot make that claim for general
   user-provided factorizations and equilibration factors if they
   differ from what ZGESVXX would itself produce.

                                                                  The␣
↪following steps are performed:

   1. If FACT = 'E', double precision scaling factors are computed to equilibrate
   the system:

     TRANS = 'N':  diag(R)*A*diag(C)     *inv(diag(C))*X = diag(R)*B
     TRANS = 'T': (diag(R)*A*diag(C))**T *inv(diag(R))*X = diag(C)*B
     TRANS = 'C': (diag(R)*A*diag(C))**H *inv(diag(R))*X = diag(C)*B

   Whether or not the system will be equilibrated depends on the
   scaling of the matrix A, but if equilibration is used, A is
   overwritten by diag(R)*A*diag(C) and B by diag(R)*B (if TRANS='N')
   or diag(C)*B (if TRANS = 'T' or 'C').

   2. If FACT = 'N' or 'E', the LU decomposition is used to factor
   the matrix A (after equilibration if FACT = 'E') as

     A = P * L * U,

   where P is a permutation matrix, L is a unit lower triangular
   matrix, and U is upper triangular.

   3. If some U(i,i)=0, so that U is exactly singular, then the
   routine returns with INFO = i. Otherwise, the factored form of A
   is used to estimate the condition number of the matrix A (see
   argument RCOND). If the reciprocal of the condition number is less
   than machine precision, the routine still goes on to solve for X
   and compute error bounds as described below.

   4. The system of equations is solved for X using the factored form
   of A.

   5. By default (unless PARAMS(LA_LINRX_ITREF_I) is set to zero),
   the routine will use iterative refinement to try to get a small
   error and error bounds.  Refinement calculates the residual to at
   least twice the working precision.

   6. If equilibration was used, the matrix X is premultiplied by
   diag(C) (if TRANS = 'N') or diag(R) (if TRANS = 'T' or 'C') so
   that it solves the original system before equilibration.
↪   Some optional parameters are bundled in the PARAMS array.  These
    settings determine how refinement is performed, but often the
    defaults are acceptable.  If the defaults are acceptable, users
    can pass NPARAMS = 0 which prevents the source code from accessing
    the PARAMS argument.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zgesvxx(FACT, TRANS, N, NRHS, A, LDA, AF, LDAF, IPIV, EQUED, R, C,
                   B, LDB, X, LDX, RCOND, RPVGRW, BERR, N_ERR_BNDS,
                   ERR_BNDS_NORM, ERR_BNDS_COMP, NPARAMS, PARAMS, WORK, RWORK,
                   INFO)
```

C specification:

```
#include "armpl.h"

void zgesvxx_(const char *fact, const char *trans, const armpl_int_t *n,
              const armpl_int_t *nrhs, armpl_doublecomplex_t *a,
              const armpl_int_t *lda, armpl_doublecomplex_t *af,
              const armpl_int_t *ldaf, armpl_int_t *ipiv, char *equed,
              double *r, double *c, armpl_doublecomplex_t *b,
              const armpl_int_t *ldb, armpl_doublecomplex_t *x,
              const armpl_int_t *ldx, double *rcond, double *rpvgrw,
              double *berr, const armpl_int_t *n_err_bnds,
              double *err_bnds_norm, double *err_bnds_comp,
              const armpl_int_t *nparams, double *params,
              armpl_doublecomplex_t *work, double *rwork, armpl_int_t *info,
              ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AF and IPIV contain the factored form of A. If EQUED is not 'N', the matrix A has been equilibrated with scaling factors given by R and C. A, AF, and IPIV are not modified. = 'N': The matrix A will be copied to AF and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AF and factored.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate Transpose)

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A. If FACT = 'F' and EQUED is not 'N', then A must have been equilibrated by the scaling factors in R and/or C. A is not modified if FACT = 'F' or 'N', or if FACT = 'E' and EQUED = 'N' on exit.

On exit, if EQUED .ne. 'N', A is scaled as follows: EQUED = 'R': A := diag(R) * A EQUED = 'C': A := A * diag(C) EQUED = 'B': A := diag(R) * A * diag(C).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input and output parameter.

AF is COMPLEX*16

AF is an array, dimension (LDAF, N). If FACT = 'F', then AF is an input argument and on entry contains the factors L and U from the factorization A = P*L*U as computed by ZGETRF. If EQUED .ne. 'N', then AF is the factored form of the equilibrated matrix A.

If FACT = 'N', then AF is an output argument and on exit returns the factors L and U from the factorization A = P*L*U of the original matrix A.

If FACT = 'E', then AF is an output argument and on exit returns the factors L and U from the factorization A = P*L*U of the equilibrated matrix A (see the description of A for the form of the equilibrated matrix).

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains the pivot indices from the factorization A = P*L*U as computed by ZGETRF; row i of the matrix was interchanged with row IPIV(i).

If FACT = 'N', then IPIV is an output argument and on exit contains the pivot indices from the factorization A = P*L*U of the original matrix A.

If FACT = 'E', then IPIV is an output argument and on exit contains the pivot indices from the factorization A = P*L*U of the equilibrated matrix A.

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'R': Row equilibration, i.e., A has been premultiplied by diag(R). = 'C': Column equilibration, i.e., A has been postmultiplied by diag(C). = 'B': Both row and column equilibration, i.e., A has been replaced by diag(R) * A * diag(C). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

R Input and output parameter.

R is DOUBLE PRECISION

R is an array, dimension (N). The row scale factors for A. If EQUED = 'R' or 'B', A is multiplied on the left by diag(R); if EQUED = 'N' or 'C', R is not accessed. R is an input argument if FACT = 'F'; otherwise, R is an output argument. If FACT = 'F' and EQUED = 'R' or 'B', each element of R must be positive. If R is output, each element of R is a power of the radix. If R is input, each element of R should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). The column scale factors for A. If EQUED = 'C' or 'B', A is multiplied on the right by diag(C); if EQUED = 'N' or 'R', C is not accessed. C is an input argument if FACT = 'F'; otherwise, C is an output argument. If FACT = 'F' and EQUED = 'C' or 'B', each element of C must be positive. If C is output, each element of C is a power of the radix. If C is input, each element of C should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause

rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if EQUED = 'N', B is not modified; if TRANS = 'N' and EQUED = 'R' or 'B', B is overwritten by diag(R)*B; if TRANS = 'T' or 'C' and EQUED = 'C' or 'B', B is overwritten by diag(C)*B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). If INFO = 0, the N-by-NRHS solution matrix X to the original system of equations. Note that A and B are modified on exit if EQUED .ne. 'N', and the solution to the equilibrated system is inv(diag(C))* X if TRANS = 'N' and EQUED = 'C' or 'B', or inv(diag(R))* X if TRANS = 'T' or 'C' and EQUED = 'R' or 'B'.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

RPVGRW Output parameter.

RPVGRW is DOUBLE PRECISION

Reciprocal pivot growth. On exit, this contains the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If this is much less than 1, then the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, estimated condition numbers, and error bounds could be unreliable. If factorization fails with 0<INFO<=N, then this contains the reciprocal pivot growth factor for the leading INFO columns of A. In ZGESVX, this quantity is returned in WORK(1).

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). Componentwise relative backward error. This is the componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

N_ERR_BNDS Input parameter.

N_ERR_BNDS is INTEGER

Number of error bounds to return for each right hand side and each type (normwise or componentwise). See ERR_BNDS_NORM and ERR_BNDS_COMP below.

ERR_BNDS_NORM Output parameter.

ERR_BNDS_NORM is DOUBLE PRECISION

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * dlamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * dlamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * dlamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Output parameter.

ERR_BNDS_COMP is DOUBLE PRECISION

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:, N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * dlamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * dlamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * dlamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

NPARAMS Input parameter.

NPARAMS is INTEGER

Specifies the number of parameters set in PARAMS. If .LE. 0, the PARAMS array is never referenced and default values are used.

PARAMS Input and output parameter.

PARAMS is DOUBLE PRECISION

PARAMS is an array, dimension NPARAMS. Specifies algorithm parameters. If an entry is .LT. 0.0, then that entry will be filled with default value used for that parameter. Only positions up to NPARAMS are accessed; defaults are used for higher-numbered parameters.

PARAMS(LA_LINRX_ITREF_I = 1) : Whether to perform iterative refinement or not. Default: 1.0D+0 = 0.0 : No refinement is performed, and no error bounds are computed. = 1.0 : Use the extra-precise refinement algorithm. (other values are reserved for future use)

PARAMS(LA_LINRX_ITHRESH_I = 2) : Maximum number of residual computations allowed for refinement. Default: 10 Aggressive: Set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in err_bnds_norm and err_bnds_comp may no longer be trustworthy.

PARAMS(LA_LINRX_CWISE_I = 3) : Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Default: 1.0 (attempt componentwise convergence)

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. The solution to every right-hand side is guaranteed. < 0: If INFO = -i, the i-th argument had an illegal value > 0 and <= N: U(INFO, INFO) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+J: The solution corresponding to the Jth right-hand side is not guaranteed. The solutions corresponding to other right- hand sides K with K > J may not be guaranteed as well, but only the first such right-hand side is reported. If a small componentwise error is not requested (PARAMS(3) = 0.0) then the Jth right-hand side is the first with a normwise error bound that is not guaranteed (the smallest J such that ERR_BNDS_NORM(J,1) = 0.0). By default (PARAMS(3) = 1.0) the Jth right-hand side is the first with either a normwise or componentwise error bound that is not guaranteed (the smallest J such that either ERR_BNDS_NORM(J,1) = 0.0 or ERR_BNDS_COMP(J,1) = 0.0). See the definition of ERR_BNDS_NORM(:,1) and ERR_BNDS_COMP(:,1). To get information about all of the right-hand sides check ERR_BNDS_NORM or ERR_BNDS_COMP.

Related Information

For this routine in other precisions, please see *cgesvxx*, *dgesvxx* and *sgesvxx*. It also exists with a native C interface as *LAPACKE_zgesvxx*.

4.6.148 zgetrs

zgetrs solves a system of linear equations

```
A * X = B,  A**T * X = B,  or  A**H * X = B
```

with a general N-by-N matrix A using the LU factorization computed by ZGETRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgetrs(TRANS, N, NRHS, A, LDA, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void zgetrs_(const char *trans, const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_int_t *ipiv, armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose)

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The factors L and U from the factorization A = P*L*U as computed by ZGETRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from ZGETRF; for 1<=i<=N, row i of the matrix was interchanged with row IPIV(i).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgetrs*, *dgetrs* and *sgetrs*. It also exists with a native C interface as *LAPACKE_zgetrs*.

4.6.149 zgtsv

zgtsv solves the equation

```
A*X = B,
```

where A is an N-by-N tridiagonal matrix, by Gaussian elimination with partial pivoting.

Note that the equation A^T *X = B may be solved by interchanging the order of the arguments DU and DL.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgtsv(N, NRHS, DL, D, DU, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void zgtsv_(const armpl_int_t *n, const armpl_int_t *nrhs,
            armpl_doublecomplex_t *dl, armpl_doublecomplex_t *d,
            armpl_doublecomplex_t *du, armpl_doublecomplex_t *b,
            const armpl_int_t *ldb, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

DL Input and output parameter.

DL is COMPLEX*16

DL is an array, dimension (N-1). On entry, DL must contain the (n-1) subdiagonal elements of A. On exit, DL is overwritten by the (n-2) elements of the second superdiagonal of the upper triangular matrix U from the LU factorization of A, in DL(1), . . . , DL(n-2).

D Input and output parameter.

D is COMPLEX*16

D is an array, dimension (N). On entry, D must contain the diagonal elements of A. On exit, D is overwritten by the n diagonal elements of U.

DU Input and output parameter.

DU is COMPLEX*16

DU is an array, dimension (N-1). On entry, DU must contain the (n-1) superdiagonal elements of A. On exit, DU is overwritten by the (n-1) elements of the first superdiagonal of U.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, U(i,i) is exactly zero, and the solution has not been computed. The factorization has not been completed unless i = N.

Related Information

For this routine in other precisions, please see *cgtsv*, *dgtsv* and *sgtsv*. It also exists with a native C interface as *LAPACKE_zgtsv*.

4.6.150 zgtsvx

zgtsvx uses the LU factorization to compute the solution to a complex system of linear equations A * X = B, A^T * X = B, or A^H * X = B, where A is a tridiagonal matrix of order N and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed:
```

1. If FACT = 'N', the LU decomposition is used to factor the matrix A

```
 as A = L * U, where L is a product of permutation and unit lower
bidiagonal matrices and U is upper triangular with nonzeros in
only the main diagonal and first two superdiagonals.
```

2. If some U(i,i)=0, so that U is exactly singular, then the routine

```
 returns with INFO = i. Otherwise, the factored form of A is used
to estimate the condition number of the matrix A.  If the
reciprocal of the condition number is less than machine precision,
INFO = N+1 is returned as a warning, but the routine still goes on
to solve for X and compute error bounds as described below.
```

3. The system of equations is solved for X using the factored form

```
of A.
```

4. Iterative refinement is applied to improve the computed solution

```
 matrix and calculate error bounds and backward error estimates
for it.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zgtsvx(FACT, TRANS, N, NRHS, DL, D, DU, DLF, DF, DUF, DU2, IPIV, B,
                  LDB, X, LDX, RCOND, FERR, BERR, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgtsvx_(const char *fact, const char *trans, const armpl_int_t *n,
             const armpl_int_t *nrhs, const armpl_doublecomplex_t *dl,
             const armpl_doublecomplex_t *d, const armpl_doublecomplex_t *du,
             armpl_doublecomplex_t *dlf, armpl_doublecomplex_t *df,
             armpl_doublecomplex_t *duf, armpl_doublecomplex_t *du2,
             armpl_int_t *ipiv, const armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, armpl_doublecomplex_t *x,
             const armpl_int_t *ldx, double *rcond, double *ferr,
             double *berr, armpl_doublecomplex_t *work, double *rwork,
             armpl_int_t *info, ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of A has been supplied on entry. = 'F': DLF, DF, DUF, DU2, and IPIV contain the factored form of A; DL, D, DU, DLF, DF, DUF, DU2 and IPIV will not be modified. = 'N': The matrix will be copied to DLF, DF, and DUF and factored.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose)

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

DL Input parameter.

DL is COMPLEX*16

DL is an array, dimension (N-1). The (n-1) subdiagonal elements of A.

D Input parameter.

D is COMPLEX*16

D is an array, dimension (N). The n diagonal elements of A.

DU Input parameter.

DU is COMPLEX*16

DU is an array, dimension (N-1). The (n-1) superdiagonal elements of A.

DLF Input and output parameter.

DLF is COMPLEX*16

DLF is an array, dimension (N-1). If FACT = 'F', then DLF is an input argument and on entry contains the (n-1) multipliers that define the matrix L from the LU factorization of A as computed by ZGTTRF.

If FACT = 'N', then DLF is an output argument and on exit contains the (n-1) multipliers that define the matrix L from the LU factorization of A.

DF Input and output parameter.

DF is COMPLEX*16

DF is an array, dimension (N). If FACT = 'F', then DF is an input argument and on entry contains the n diagonal elements of the upper triangular matrix U from the LU factorization of A.

If FACT = 'N', then DF is an output argument and on exit contains the n diagonal elements of the upper triangular matrix U from the LU factorization of A.

DUF Input and output parameter.

DUF is COMPLEX*16

DUF is an array, dimension (N-1). If FACT = 'F', then DUF is an input argument and on entry contains the (n-1) elements of the first superdiagonal of U.

If FACT = 'N', then DUF is an output argument and on exit contains the (n-1) elements of the first superdiagonal of U.

DU2 Input and output parameter.

DU2 is COMPLEX*16

DU2 is an array, dimension (N-2). If FACT = 'F', then DU2 is an input argument and on entry contains the (n-2) elements of the second superdiagonal of U.

If FACT = 'N', then DU2 is an output argument and on exit contains the (n-2) elements of the second superdiagonal of U.

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains the pivot indices from the LU factorization of A as computed by ZGTTRF.

If FACT = 'N', then IPIV is an output argument and on exit contains the pivot indices from the LU factorization of A; row i of the matrix was interchanged with row IPIV(i). IPIV(i) will always be either i or i+1; IPIV(i) = i indicates a row interchange was not required.

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). The N-by-NRHS right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The estimate of the reciprocal condition number of the matrix A. If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: U(i,i) is exactly zero. The factorization has not been completed unless i = N, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+1: U is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *cgtsvx*, *dgtsvx* and *sgtsvx*. It also exists with a native C interface as *LAPACKE_zgtsvx*.

4.6.151 zgttrs

zgttrs solves one of the systems of equations

```
A * X = B,  A**T * X = B,  or  A**H * X = B,
```

with a tridiagonal matrix A using the LU factorization computed by ZGTTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgttrs(TRANS, N, NRHS, DL, D, DU, DU2, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void zgttrs_(const char *trans, const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_doublecomplex_t *dl, const armpl_doublecomplex_t *d,
             const armpl_doublecomplex_t *du,
             const armpl_doublecomplex_t *du2, const armpl_int_t *ipiv,
             armpl_doublecomplex_t *b, const armpl_int_t *ldb,
             armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations. = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose)

N Input parameter.

N is INTEGER

The order of the matrix A.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

DL Input parameter.

DL is COMPLEX*16

DL is an array, dimension (N-1). The (n-1) multipliers that define the matrix L from the LU factorization of A.

D Input parameter.

D is COMPLEX*16

D is an array, dimension (N). The n diagonal elements of the upper triangular matrix U from the LU factorization of A.

DU Input parameter.

DU is COMPLEX*16

DU is an array, dimension (N-1). The (n-1) elements of the first super-diagonal of U.

DU2 Input parameter.

DU2 is COMPLEX*16

DU2 is an array, dimension (N-2). The (n-2) elements of the second super-diagonal of U.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices; for 1 <= i <= n, row i of the matrix was interchanged with row IPIV(i). IPIV(i) will always be either i or i+1; IPIV(i) = i indicates a row interchange was not required.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the matrix of right hand side vectors B. On exit, B is overwritten by the solution vectors X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgttrs*, *dgttrs* and *sgttrs*. It also exists with a native C interface as *LAPACKE_zgttrs*.

4.6.152 zhesv

zhesv computes the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N Hermitian matrix and X and B are N-by-NRHS matrices.

The diagonal pivoting method is used to factor A as

```
A = U * D * U**H,  if UPLO = 'U', or
A = L * D * L**H,  if UPLO = 'L',
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is Hermitian and block diagonal with 1-by-1 and 2-by-2 diagonal blocks. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhesv(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhesv_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
            armpl_doublecomplex_t *a, const armpl_int_t *lda,
            armpl_int_t *ipiv, armpl_doublecomplex_t *b,
            const armpl_int_t *ldb, armpl_doublecomplex_t *work,
            const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^H or A = L*D*L^H as computed by ZHETRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D, as determined by ZHETRF. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged, and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= 1, and for best performance LWORK >= max(1,N*NB), where NB is the optimal blocksize for ZHETRF. for LWORK < N, TRS will be done with Level BLAS 2 for LWORK >= N, TRS will be done with Level BLAS 3

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, so the solution could not be computed.

Related Information

For this routine in other precisions, please see *chesv*. It also exists with a native C interface as *LAPACKE_zhesv*.

4.6.153 zhesv_aa

ZHESV_AA computes the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N Hermitian matrix and X and B are N-by-NRHS matrices.

Aasen's algorithm is used to factor A as

```
A = U * T * U**H,  if UPLO = 'U', or
A = L * T * L**H,  if UPLO = 'L',
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and T is Hermitian and tridiagonal. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhesv_aa(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhesv_aa_(const char *uplo, const armpl_int_t *n,
               const armpl_int_t *nrhs, armpl_doublecomplex_t *a,
               const armpl_int_t *lda, armpl_int_t *ipiv,
```

(continues on next page)

(continued from previous page)

```
                armpl_doublecomplex_t *b, const armpl_int_t *ldb,
                armpl_doublecomplex_t *work, const armpl_int_t *lwork,
                armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the tridiagonal matrix T and the multipliers used to obtain the factor U or L from the factorization A = U*T*U^H or A = L*T*L^H as computed by ZHETRF_AA.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

On exit, it contains the details of the interchanges, i.e., the row and column k of A were interchanged with the row and column IPIV(k).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= MAX(1,2*N,3*N-2), and for best performance LWORK >= max(1,N*NB), where NB is the optimal blocksize for ZHETRF.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, so the solution could not be computed.

Related Information

For this routine in other precisions, please see *chesv_aa*. It also exists with a native C interface as *LAPACKE_zhesv_aa*.

4.6.154 zhesv_rk

ZHESV_RK computes the solution to a complex system of linear equations A * X = B, where A is an N-by-N Hermitian matrix and X and B are N-by-NRHS matrices.

The bounded Bunch-Kaufman (rook) diagonal pivoting method is used to factor A as

```
A = P*U*D*(U**H)*(P**T),  if UPLO = 'U', or
A = P*L*D*(L**H)*(P**T),  if UPLO = 'L',
```

where U (or L) is unit upper (or lower) triangular matrix, U^H (or L^H) is the conjugate of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is Hermitian and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

ZHETRF_RK is called to compute the factorization of a complex Hermitian matrix. The factored form of A is then used to solve the system of equations A * X = B by calling BLAS3 routine ZHETRS_3.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhesv_rk(UPLO, N, NRHS, A, LDA, E, IPIV, B, LDB, WORK, LWORK,
                    INFO)
```

C specification:

```
#include "armpl.h"

void zhesv_rk_(const char *uplo, const armpl_int_t *n,
               const armpl_int_t *nrhs, armpl_doublecomplex_t *a,
               const armpl_int_t *lda, armpl_doublecomplex_t *e,
               armpl_int_t *ipiv, armpl_doublecomplex_t *b,
               const armpl_int_t *ldb, armpl_doublecomplex_t *work,
               const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the Hermitian matrix A is stored: = 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U': the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced.

If UPLO = 'L': the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, diagonal of the block diagonal matrix D and factors U or L as computed by ZHETRF_RK: a) ONLY diagonal elements of the Hermitian block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

For more info see the description of ZHETRF_RK routine.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Output parameter.

E is COMPLEX*16

E is an array, dimension (N). On exit, contains the output computed by the factorization routine ZHETRF_RK, i.e. the superdiagonal (or subdiagonal) elements of the Hermitian block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) is set to 0; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) is set to 0.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is set to 0 in both UPLO = 'U' or UPLO = 'L' cases.

For more info see the description of ZHETRF_RK routine.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D, as determined by ZHETRF_RK.

For more info see the description of ZHETRF_RK routine.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)).. Work array used in the factorization stage. On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= 1. For best performance of factorization stage LWORK >= max(1,N*NB), where NB is the optimal blocksize for ZHETRF_RK.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array for factorization stage, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit

< 0: If INFO = -k, the k-th argument had an illegal value

> 0: If INFO = k, the matrix A is singular, because: If UPLO = 'U': column k in the upper triangular part of A contains all zeros. If UPLO = 'L': column k in the lower triangular part of A contains all zeros.

Therefore D(k,k) is exactly zero, and superdiagonal elements of column k of U (or subdiagonal elements of column k of L) are all zeros. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

NOTE: INFO only stores the first occurrence of a singularity, any subsequent occurrence of singularity is not stored in INFO even though the factorization always completes.

Related Information

For this routine in other precisions, please see *chesv_rk*. It also exists with a native C interface as *LAPACKE_zhesv_rk*.

4.6.155 zhesv_rook

ZHESV_ROOK computes the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N Hermitian matrix and X and B are N-by-NRHS matrices.

The bounded Bunch-Kaufman ("rook") diagonal pivoting method is used to factor A as

```
 A = U * D * U**T,  if UPLO = 'U', or
A = L * D * L**T,  if UPLO = 'L',
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is Hermitian and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

ZHETRF_ROOK is called to compute the factorization of a complex Hermition matrix A using the bounded Bunch-Kaufman ("rook") diagonal pivoting method.

The factored form of A is then used to solve the system of equations A * X = B by calling ZHETRS_ROOK (uses BLAS 2).

Syntax

Fortran specification:

```
use armpl_library

subroutine zhesv_rook(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhesv_rook_(const char *uplo, const armpl_int_t *n,
                 const armpl_int_t *nrhs, armpl_doublecomplex_t *a,
                 const armpl_int_t *lda, armpl_int_t *ipiv,
                 armpl_doublecomplex_t *b, const armpl_int_t *ldb,
                 armpl_doublecomplex_t *work, const armpl_int_t *lwork,
                 armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^H or A = L*D*L^H as computed by ZHETRF_ROOK.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D.

If UPLO = 'U': Only the last KB elements of IPIV are set.

If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k-1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k-1 and -IPIV(k-1) were inerchaged, D(k-1:k,k-1:k) is a 2-by-2 diagonal block.

If UPLO = 'L': Only the first KB elements of IPIV are set.

If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k+1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k+1 and -IPIV(k+1) were inerchaged, D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= 1, and for best performance LWORK >= max(1,N*NB), where NB is the optimal blocksize for ZHETRF_ROOK. for LWORK < N, TRS will be done with Level BLAS 2 for LWORK >= N, TRS will be done with Level BLAS 3

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, so the solution could not be computed.

Related Information

For this routine in other precisions, please see *chesv_rook*.

4.6.156 zhesvx

zhesvx uses the diagonal pivoting factorization to compute the solution to a complex system of linear equations A * X = B, where A is an N-by-N Hermitian matrix and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed:
```

1. If FACT = 'N', the diagonal pivoting method is used to factor A.

```
 The form of the factorization is
   A = U * D * U**H,  if UPLO = 'U', or
   A = L * D * L**H,  if UPLO = 'L',
where U (or L) is a product of permutation and unit upper (lower)
triangular matrices, and D is Hermitian and block diagonal with
1-by-1 and 2-by-2 diagonal blocks.
```

2. If some D(i,i)=0, so that D is exactly singular, then the routine

```
 returns with INFO = i. Otherwise, the factored form of A is used
to estimate the condition number of the matrix A.  If the
reciprocal of the condition number is less than machine precision,
INFO = N+1 is returned as a warning, but the routine still goes on
to solve for X and compute error bounds as described below.
```

3. The system of equations is solved for X using the factored form

```
of A.
```

4. Iterative refinement is applied to improve the computed solution

```
 matrix and calculate error bounds and backward error estimates
for it.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zhesvx(FACT, UPLO, N, NRHS, A, LDA, AF, LDAF, IPIV, B, LDB, X, LDX,
                  RCOND, FERR, BERR, WORK, LWORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhesvx_(const char *fact, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *nrhs, const armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_doublecomplex_t *af,
             const armpl_int_t *ldaf, armpl_int_t *ipiv,
             const armpl_doublecomplex_t *b, const armpl_int_t *ldb,
             armpl_doublecomplex_t *x, const armpl_int_t *ldx, double *rcond,
             double *ferr, double *berr, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, double *rwork, armpl_int_t *info,
             ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of A has been supplied on entry. = 'F': On entry, AF and IPIV contain the factored form of A. A, AF and IPIV will not be modified. = 'N': The matrix A will be copied to AF and factored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input and output parameter.

AF is COMPLEX*16

AF is an array, dimension (LDAF, N). If FACT = 'F', then AF is an input argument and on entry contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^H or A = L*D*L^H as computed by ZHETRF.

If FACT = 'N', then AF is an output argument and on exit returns the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^H or A = L*D*L^H .

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains details of the interchanges and the block structure of D, as determined by ZHETRF. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

If FACT = 'N', then IPIV is an output argument and on exit contains details of the interchanges and the block structure of D, as determined by ZHETRF.

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). The N-by-NRHS right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The estimate of the reciprocal condition number of the matrix A. If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= max(1,2*N), and for best performance, when FACT = 'N', LWORK >= max(1,2*N,N*NB), where NB is the optimal blocksize for ZHETRF.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: D(i,i) is exactly zero. The factorization has been completed but the factor D is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+1: D is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *chesvx*. It also exists with a native C interface as *LAPACKE_zhesvx*.

4.6.157 zhesvxx

```
ZHESVXX uses the diagonal pivoting factorization to compute the
   solution to a complex*16 system of linear equations A * X = B, where
   A is an N-by-N symmetric matrix and X and B are N-by-NRHS
   matrices.

   If requested, both normwise and maximum componentwise error bounds
   are returned. ZHESVXX will return a solution with a tiny
   guaranteed error (O(eps) where eps is the working machine
   precision) unless the matrix is very ill-conditioned, in which
   case a warning is returned. Relevant condition numbers also are
   calculated and returned.

   ZHESVXX accepts user-provided factorizations and equilibration
   factors; see the definitions of the FACT and EQUED options.
   Solving with refinement and using a factorization from a previous
   ZHESVXX call will also produce a solution with either O(eps)
   errors or warnings, but we cannot make that claim for general
   user-provided factorizations and equilibration factors if they
   differ from what ZHESVXX would itself produce.

                                                                  The␣
↪following steps are performed:

   1. If FACT = 'E', double precision scaling factors are computed to equilibrate
   the system:

     diag(S)*A*diag(S)     *inv(diag(S))*X = diag(S)*B

   Whether or not the system will be equilibrated depends on the
   scaling of the matrix A, but if equilibration is used, A is
   overwritten by diag(S)*A*diag(S) and B by diag(S)*B.

   2. If FACT = 'N' or 'E', the LU decomposition is used to factor
   the matrix A (after equilibration if FACT = 'E') as

      A = U * D * U**T,  if UPLO = 'U', or
      A = L * D * L**T,  if UPLO = 'L',

   where U (or L) is a product of permutation and unit upper (lower)
   triangular matrices, and D is symmetric and block diagonal with
   1-by-1 and 2-by-2 diagonal blocks.

   3. If some D(i,i)=0, so that D is exactly singular, then the
   routine returns with INFO = i. Otherwise, the factored form of A
   is used to estimate the condition number of the matrix A (see
   argument RCOND).  If the reciprocal of the condition number is
   less than machine precision, the routine still goes on to solve
   for X and compute error bounds as described below.

   4. The system of equations is solved for X using the factored form
   of A.

   5. By default (unless PARAMS(LA_LINRX_ITREF_I) is set to zero),
   the routine will use iterative refinement to try to get a small
   error and error bounds.  Refinement calculates the residual to at
```

(continues on next page)

(continued from previous page)

```
   least twice the working precision.

   6. If equilibration was used, the matrix X is premultiplied by
   diag(R) so that it solves the original system before
   equilibration.                          Some optional parameters are bundled␣
↪in the PARAMS array.  These
    settings determine how refinement is performed, but often the
    defaults are acceptable.  If the defaults are acceptable, users
    can pass NPARAMS = 0 which prevents the source code from accessing
    the PARAMS argument.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zhesvxx(FACT, UPLO, N, NRHS, A, LDA, AF, LDAF, IPIV, EQUED, S, B,
                   LDB, X, LDX, RCOND, RPVGRW, BERR, N_ERR_BNDS,
                   ERR_BNDS_NORM, ERR_BNDS_COMP, NPARAMS, PARAMS, WORK, RWORK,
                   INFO)
```

C specification:

```
#include "armpl.h"

void zhesvxx_(const char *fact, char *uplo, const armpl_int_t *n,
              const armpl_int_t *nrhs, armpl_doublecomplex_t *a,
              const armpl_int_t *lda, armpl_doublecomplex_t *af,
              const armpl_int_t *ldaf, armpl_int_t *ipiv, char *equed,
              double *s, armpl_doublecomplex_t *b, const armpl_int_t *ldb,
              armpl_doublecomplex_t *x, const armpl_int_t *ldx, double *rcond,
              double *rpvgrw, double *berr, const armpl_int_t *n_err_bnds,
              double *err_bnds_norm, double *err_bnds_comp,
              const armpl_int_t *nparams, double *params,
              armpl_doublecomplex_t *work, double *rwork, armpl_int_t *info,
              ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AF and IPIV contain the factored form of A. If EQUED is not 'N', the matrix A has been equilibrated with scaling factors given by S. A, AF, and IPIV are not modified. = 'N': The matrix A will be copied to AF and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AF and factored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if FACT = 'E' and EQUED = 'Y', A is overwritten by diag(S)*A*diag(S).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input and output parameter.

AF is COMPLEX*16

AF is an array, dimension (LDAF, N). If FACT = 'F', then AF is an input argument and on entry contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by DSYTRF.

If FACT = 'N', then AF is an output argument and on exit returns the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T .

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains details of the interchanges and the block structure of D, as determined by ZHETRF. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

If FACT = 'N', then IPIV is an output argument and on exit contains details of the interchanges and the block structure of D, as determined by ZHETRF.

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'Y': Both row and column equilibration, i.e., A has been replaced by diag(S) * A * diag(S). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

S Input and output parameter.

S is DOUBLE PRECISION

S is an array, dimension (N). The scale factors for A. If EQUED = 'Y', A is multiplied on the left and right by diag(S). S is an input argument if FACT = 'F'; otherwise, S is an output argument. If FACT = 'F' and EQUED = 'Y', each element of S must be positive. If S is output, each element of S is a power of the radix. If S is input, each element of S should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or

overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if EQUED = 'N', B is not modified; if EQUED = 'Y', B is overwritten by diag(S)*B;

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). If INFO = 0, the N-by-NRHS solution matrix X to the original system of equations. Note that A and B are modified on exit if EQUED .ne. 'N', and the solution to the equilibrated system is inv(diag(S))*X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

RPVGRW Output parameter.

RPVGRW is DOUBLE PRECISION

Reciprocal pivot growth. On exit, this contains the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If this is much less than 1, then the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, estimated condition numbers, and error bounds could be unreliable. If factorization fails with 0<INFO<=N, then this contains the reciprocal pivot growth factor for the leading INFO columns of A.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). Componentwise relative backward error. This is the componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

N_ERR_BNDS Input parameter.

N_ERR_BNDS is INTEGER

Number of error bounds to return for each right hand side and each type (normwise or componentwise). See ERR_BNDS_NORM and ERR_BNDS_COMP below.

ERR_BNDS_NORM Output parameter.

ERR_BNDS_NORM is DOUBLE PRECISION

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * dlamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * dlamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * dlamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Output parameter.

ERR_BNDS_COMP is DOUBLE PRECISION

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:, N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * dlamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * dlamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * dlamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

NPARAMS Input parameter.

NPARAMS is INTEGER

Specifies the number of parameters set in PARAMS. If .LE. 0, the PARAMS array is never referenced and default values are used.

PARAMS Input and output parameter.

PARAMS is DOUBLE PRECISION

PARAMS is an array, dimension NPARAMS. Specifies algorithm parameters. If an entry is .LT. 0.0, then that entry will be filled with default value used for that parameter. Only positions up to NPARAMS are accessed; defaults are used for higher-numbered parameters.

PARAMS(LA_LINRX_ITREF_I = 1) : Whether to perform iterative refinement or not. Default: 1.0D+0 = 0.0 : No refinement is performed, and no error bounds are computed. = 1.0 : Use the extra-precise refinement algorithm. (other values are reserved for future use)

PARAMS(LA_LINRX_ITHRESH_I = 2) : Maximum number of residual computations allowed for refinement. Default: 10 Aggressive: Set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in err_bnds_norm and err_bnds_comp may no longer be trustworthy.

PARAMS(LA_LINRX_CWISE_I = 3) : Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Default: 1.0 (attempt componentwise convergence)

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (5*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. The solution to every right-hand side is guaranteed. < 0: If INFO = -i, the i-th argument had an illegal value > 0 and <= N: U(INFO, INFO) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+J: The solution corresponding to the Jth right-hand side is not guaranteed. The solutions corresponding to other right- hand sides K with K > J may not be guaranteed as well, but only the first such right-hand side is reported. If a small componentwise error is not requested (PARAMS(3) = 0.0) then the Jth right-hand side is the first with a normwise error bound that is not guaranteed (the smallest J such that ERR_BNDS_NORM(J,1) = 0.0). By default (PARAMS(3) = 1.0) the Jth right-hand side is the first with either a normwise or componentwise error bound that is not guaranteed (the smallest J such that either ERR_BNDS_NORM(J,1) = 0.0 or ERR_BNDS_COMP(J,1) = 0.0). See the definition of ERR_BNDS_NORM(:,1) and ERR_BNDS_COMP(:,1). To get information about all of the right-hand sides check ERR_BNDS_NORM or ERR_BNDS_COMP.

Related Information

For this routine in other precisions, please see *chesvxx*. It also exists with a native C interface as *LAPACKE_zhesvxx*.

4.6.158 zhetrs

zhetrs solves a system of linear equations A*X = B with a complex Hermitian matrix A using the factorization A = U*D*U^H or A = L*D*L^H computed by ZHETRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhetrs(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void zhetrs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_int_t *ipiv, armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^H ; = 'L': Lower triangular, form is A = L*D*L^H .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by ZHETRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZHETRF.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *chetrs*. It also exists with a native C interface as *LAPACKE_zhetrs*.

4.6.159 zhetrs2

zhetrs2 solves a system of linear equations A*X = B with a complex Hermitian matrix A using the factorization A = U*D*U^H or A = L*D*L^H computed by ZHETRF and converted by ZSYCONV.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhetrs2(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhetrs2_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
              const armpl_doublecomplex_t *a, const armpl_int_t *lda,
              const armpl_int_t *ipiv, armpl_doublecomplex_t *b,
              const armpl_int_t *ldb, armpl_doublecomplex_t *work,
              armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^H ; = 'L': Lower triangular, form is A = L*D*L^H .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by ZHETRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZHETRF.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *chetrs2*. It also exists with a native C interface as *LAPACKE_zhetrs2*.

4.6.160 zhetrs_3

ZHETRS_3 solves a system of linear equations A * X = B with a complex Hermitian matrix A using the factorization computed by ZHETRF_RK or ZHETRF_BK:

```
A = P*U*D*(U**H)*(P**T) or A = P*L*D*(L**H)*(P**T),
```

where U (or L) is unit upper (or lower) triangular matrix, U^H (or L^H) is the conjugate of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is Hermitian and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This algorithm is using Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhetrs_3(UPLO, N, NRHS, A, LDA, E, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void zhetrs_3_(const char *uplo, const armpl_int_t *n,
               const armpl_int_t *nrhs, const armpl_doublecomplex_t *a,
               const armpl_int_t *lda, const armpl_doublecomplex_t *e,
               const armpl_int_t *ipiv, armpl_doublecomplex_t *b,
               const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix: = 'U': Upper triangular, form is A = P*U*D*(U^H)*(P^T); = 'L': Lower triangular, form is A = P*L*D*(L^H)*(P^T).

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). Diagonal of the block diagonal matrix D and factors U or L as computed by ZHETRF_RK and ZHETRF_BK: a) ONLY diagonal elements of the Hermitian block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D should be provided on entry in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Input parameter.

E is COMPLEX*16

E is an array, dimension (N). On entry, contains the superdiagonal (or subdiagonal) elements of the Hermitian block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i),i=2:N, E(1) not referenced; If UPLO = 'L': E(i) = D(i+1,i),i=1:N-1, E(N) not referenced.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is not referenced in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZHETRF_RK or ZHETRF_BK.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *chetrs_3*. It also exists with a native C interface as *LAPACKE_zhetrs_3*.

4.6.161 zhetrs_aa

ZHETRS_AA solves a system of linear equations A*X = B with a complex hermitian matrix A using the factorization A = U*T*U^H or A = L*T*L^T computed by ZHETRF_AA.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhetrs_aa(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhetrs_aa_(const char *uplo, const armpl_int_t *n,
                const armpl_int_t *nrhs, const armpl_doublecomplex_t *a,
                const armpl_int_t *lda, const armpl_int_t *ipiv,
                armpl_doublecomplex_t *b, const armpl_int_t *ldb,
                const armpl_doublecomplex_t *work, const armpl_int_t *lwork,
                armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*T*U^H ; = 'L': Lower triangular, form is A = L*T*L^H .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). Details of factors computed by ZHETRF_AA.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges as computed by ZHETRF_AA.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Input parameter.

WORK is DOUBLE array, dimension (MAX(1, LWORK))

LWORK Input parameter.

LWORK is INTEGER, LWORK >= MAX(1,3*N-2).

param[out] INFO verbatim INFO is INTEGER = 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *chetrs_aa*. It also exists with a native C interface as *LAPACKE_zhetrs_aa*.

4.6.162 zhetrs_rook

ZHETRS_ROOK solves a system of linear equations A*X = B with a complex Hermitian matrix A using the factorization A = U*D*U^H or A = L*D*L^H computed by ZHETRF_ROOK.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhetrs_rook(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void zhetrs_rook_(const char *uplo, const armpl_int_t *n,
                  const armpl_int_t *nrhs, const armpl_doublecomplex_t *a,
                  const armpl_int_t *lda, const armpl_int_t *ipiv,
                  armpl_doublecomplex_t *b, const armpl_int_t *ldb,
                  armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^H ; = 'L': Lower triangular, form is A = L*D*L^H .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by ZHETRF_ROOK.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZHETRF_ROOK.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *chetrs_rook*. It also exists with a native C interface as *LAPACKE_zhetrs_rook*.

4.6.163 zhpsv

zhpsv computes the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N Hermitian matrix stored in packed format and X and B are N-by-NRHS matrices.

The diagonal pivoting method is used to factor A as

```
A = U * D * U**H,  if UPLO = 'U', or
A = L * D * L**H,  if UPLO = 'L',
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, D is Hermitian and block diagonal with 1-by-1 and 2-by-2 diagonal blocks. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhpsv(UPLO, N, NRHS, AP, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void zhpsv_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
            armpl_doublecomplex_t *ap, armpl_int_t *ipiv,
            armpl_doublecomplex_t *b, const armpl_int_t *ldb,
            armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AP Input and output parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the Hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n. See below for further details.

On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^H or A = L*D*L^H as computed by ZHPTRF, stored as a packed triangular matrix in the same storage format as A.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D, as determined by ZHPTRF. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged, and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, so the solution could not be computed.

Related Information

For this routine in other precisions, please see *chpsv*. It also exists with a native C interface as *LAPACKE_zhpsv*.

4.6.164 zhpsvx

zhpsvx uses the diagonal pivoting factorization A = U*D*U^H or A = L*D*L^H to compute the solution to a complex system of linear equations A * X = B, where A is an N-by-N Hermitian matrix stored in packed format and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed:
```

1. If FACT = 'N', the diagonal pivoting method is used to factor A as

```
   A = U * D * U**H,  if UPLO = 'U', or
   A = L * D * L**H,  if UPLO = 'L',
where U (or L) is a product of permutation and unit upper (lower)
triangular matrices and D is Hermitian and block diagonal with
1-by-1 and 2-by-2 diagonal blocks.
```

2. If some D(i,i)=0, so that D is exactly singular, then the routine

```
 returns with INFO = i. Otherwise, the factored form of A is used
to estimate the condition number of the matrix A.  If the
reciprocal of the condition number is less than machine precision,
INFO = N+1 is returned as a warning, but the routine still goes on
to solve for X and compute error bounds as described below.
```

3. The system of equations is solved for X using the factored form

```
of A.
```

4. Iterative refinement is applied to improve the computed solution

```
 matrix and calculate error bounds and backward error estimates
for it.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zhpsvx(FACT, UPLO, N, NRHS, AP, AFP, IPIV, B, LDB, X, LDX, RCOND,
                  FERR, BERR, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhpsvx_(const char *fact, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *nrhs, const armpl_doublecomplex_t *ap,
             armpl_doublecomplex_t *afp, armpl_int_t *ipiv,
             const armpl_doublecomplex_t *b, const armpl_int_t *ldb,
             armpl_doublecomplex_t *x, const armpl_int_t *ldx, double *rcond,
             double *ferr, double *berr, armpl_doublecomplex_t *work,
             double *rwork, armpl_int_t *info, ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of A has been supplied on entry. = 'F': On entry, AFP and IPIV contain the factored form of A. AFP and IPIV will not be modified. = 'N': The matrix A will be copied to AFP and factored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AP Input parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). The upper or lower triangle of the Hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n. See below for further details.

AFP Input and output parameter.

AFP is COMPLEX*16

AFP is an array, dimension (N*(N+1)/2). If FACT = 'F', then AFP is an input argument and on entry contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^H or A = L*D*L^H as computed by ZHPTRF, stored as a packed triangular matrix in the same storage format as A.

If FACT = 'N', then AFP is an output argument and on exit contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^H or A = L*D*L^H as computed by ZHPTRF, stored as a packed triangular matrix in the same storage format as A.

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains details of the interchanges and the block structure of D, as determined by ZHPTRF. If IPIV(k) > 0, then rows and columns k and IPIV(k)

were interchanged and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

If FACT = 'N', then IPIV is an output argument and on exit contains details of the interchanges and the block structure of D, as determined by ZHPTRF.

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). The N-by-NRHS right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The estimate of the reciprocal condition number of the matrix A. If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: D(i,i) is exactly zero. The factorization has been completed but the factor D is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+1: D is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *chpsvx*. It also exists with a native C interface as *LAPACKE_zhpsvx*.

4.6.165 zpbsv

zpbsv computes the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N Hermitian positive definite band matrix and X and B are N-by-NRHS matrices.

The Cholesky decomposition is used to factor A as

```
A = U**H * U,  if UPLO = 'U', or
A = L * L**H,  if UPLO = 'L',
```

where U is an upper triangular band matrix, and L is a lower triangular band matrix, with the same number of superdiagonals or subdiagonals as A. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine zpbsv(UPLO, N, KD, NRHS, AB, LDAB, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void zpbsv_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
            const armpl_int_t *nrhs, armpl_doublecomplex_t *ab,
            const armpl_int_t *ldab, armpl_doublecomplex_t *b,
            const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AB Input and output parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the Hermitian band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(KD+1+i-j,j) = A(i,j) for max(1,j-KD)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(N,j+KD). See below for further details.

On exit, if INFO = 0, the triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H of the band matrix A, in the same storage format as A.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

Related Information

For this routine in other precisions, please see *cpbsv*, *dpbsv* and *spbsv*. It also exists with a native C interface as *LAPACKE_zpbsv*.

4.6.166 zpbsvx

zpbsvx uses the Cholesky factorization A = U^H *U or A = L*L^H to compute the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N Hermitian positive definite band matrix and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed:
```

1. If FACT = 'E', real scaling factors are computed to equilibrate

```
 the system:
   diag(S) * A * diag(S) * inv(diag(S)) * X = diag(S) * B
Whether or not the system will be equilibrated depends on the
scaling of the matrix A, but if equilibration is used, A is
overwritten by diag(S)*A*diag(S) and B by diag(S)*B.
```

2. If FACT = 'N' or 'E', the Cholesky decomposition is used to

```
 factor the matrix A (after equilibration if FACT = 'E') as
   A = U**H * U,  if UPLO = 'U', or
   A = L * L**H,  if UPLO = 'L',
where U is an upper triangular band matrix, and L is a lower
triangular band matrix.
```

3. If the leading i-by-i principal minor is not positive definite,

```
 then the routine returns with INFO = i. Otherwise, the factored
form of A is used to estimate the condition number of the matrix
A.  If the reciprocal of the condition number is less than machine
precision, INFO = N+1 is returned as a warning, but the routine
still goes on to solve for X and compute error bounds as
described below.
```

4. The system of equations is solved for X using the factored form

```
of A.
```

5. Iterative refinement is applied to improve the computed solution

```
 matrix and calculate error bounds and backward error estimates
for it.
```

6. If equilibration was used, the matrix X is premultiplied by

```
 diag(S) so that it solves the original system before
equilibration.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zpbsvx(FACT, UPLO, N, KD, NRHS, AB, LDAB, AFB, LDAFB, EQUED, S, B,
                  LDB, X, LDX, RCOND, FERR, BERR, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zpbsvx_(const char *fact, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *kd, const armpl_int_t *nrhs,
             armpl_doublecomplex_t *ab, const armpl_int_t *ldab,
             armpl_doublecomplex_t *afb, const armpl_int_t *ldafb,
             char *equed, double *s, armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, armpl_doublecomplex_t *x,
```

(continues on next page)

(continued from previous page)

```
            const armpl_int_t *ldx, double *rcond, double *ferr,
            double *berr, armpl_doublecomplex_t *work, double *rwork,
            armpl_int_t *info, ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AFB contains the factored form of A. If EQUED = 'Y', the matrix A has been equilibrated with scaling factors given by S. AB and AFB will not be modified. = 'N': The matrix A will be copied to AFB and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AFB and factored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right-hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AB Input and output parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the Hermitian band matrix A, stored in the first KD+1 rows of the array, except if FACT = 'F' and EQUED = 'Y', then A must contain the equilibrated matrix diag(S)*A*diag(S). The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(KD+1+i-j,j) = A(i,j) for max(1,j-KD)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(N,j+KD). See below for further details.

On exit, if FACT = 'E' and EQUED = 'Y', A is overwritten by diag(S)*A*diag(S).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array A. LDAB >= KD+1.

AFB Input and output parameter.

AFB is COMPLEX*16

AFB is an array, dimension (LDAFB, N). If FACT = 'F', then AFB is an input argument and on entry contains the triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H of the band matrix A, in the same storage format as A (see AB). If EQUED = 'Y', then AFB is the factored form of the equilibrated matrix A.

If FACT = 'N', then AFB is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H .

If FACT = 'E', then AFB is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H of the equilibrated matrix A (see the description of A for the form of the equilibrated matrix).

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AFB. LDAFB >= KD+1.

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'Y': Equilibration was done, i.e., A has been replaced by diag(S) * A * diag(S). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

S Input and output parameter.

S is DOUBLE PRECISION

S is an array, dimension (N). The scale factors for A; not accessed if EQUED = 'N'. S is an input argument if FACT = 'F'; otherwise, S is an output argument. If FACT = 'F' and EQUED = 'Y', each element of S must be positive.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if EQUED = 'N', B is not modified; if EQUED = 'Y', B is overwritten by diag(S) * B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X to the original system of equations. Note that if EQUED = 'Y', A and B are modified on exit, and the solution to the equilibrated system is inv(diag(S))*X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The estimate of the reciprocal condition number of the matrix A after equilibration (if done). If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude

of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed. RCOND = 0 is returned. = N+1: U is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *cpbsvx*, *dpbsvx* and *spbsvx*. It also exists with a native C interface as *LAPACKE_zpbsvx*.

4.6.167 zpbtrs

zpbtrs solves a system of linear equations A*X = B with a Hermitian positive definite band matrix A using the Cholesky factorization A = U^H *U or A = L*L^H computed by ZPBTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine zpbtrs(UPLO, N, KD, NRHS, AB, LDAB, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void zpbtrs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
             const armpl_int_t *nrhs, const armpl_doublecomplex_t *ab,
             const armpl_int_t *ldab, armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangular factor stored in AB; = 'L': Lower triangular factor stored in AB.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AB Input parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). The triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H of the band matrix A, stored in the first KD+1 rows of the array. The j-th column of U or L is stored in the j-th column of the array AB as follows: if UPLO ='U', AB(kd+1+i-j,j) = U(i,j) for max(1,j-kd)<=i<=j; if UPLO ='L', AB(1+i-j,j) = L(i,j) for j<=i<=min(n,j+kd).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cpbtrs*, *dpbtrs* and *spbtrs*. It also exists with a native C interface as *LAPACKE_zpbtrs*.

4.6.168 zpftrs

zpftrs solves a system of linear equations A*X = B with a Hermitian positive definite matrix A using the Cholesky factorization A = U^H *U or A = L*L^H computed by ZPFTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine zpftrs(TRANSR, UPLO, N, NRHS, A, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void zpftrs_(const char *transr, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *nrhs, const armpl_doublecomplex_t *a,
             armpl_doublecomplex_t *b, const armpl_int_t *ldb,
             armpl_int_t *info, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': The Normal TRANSR of RFP A is stored; = 'C': The Conjugate-transpose TRANSR of RFP A is stored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of RFP A is stored; = 'L': Lower triangle of RFP A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (N*(N+1)/2);. The triangular factor U or L from the Cholesky factorization of RFP A = U^H *U or RFP A = L*L^H , as computed by ZPFTRF. See note below for more details about RFP A.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cpftrs*, *dpftrs* and *spftrs*. It also exists with a native C interface as *LAPACKE_zpftrs*.

4.6.169 zposv

zposv computes the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N Hermitian positive definite matrix and X and B are N-by-NRHS matrices.

The Cholesky decomposition is used to factor A as

```
A = U**H* U,  if UPLO = 'U', or
A = L * L**H,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is a lower triangular matrix. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine zposv(UPLO, N, NRHS, A, LDA, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void zposv_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
            armpl_doublecomplex_t *a, const armpl_int_t *lda,
            armpl_doublecomplex_t *b, const armpl_int_t *ldb,
            armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H .

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

Related Information

For this routine in other precisions, please see *cposv*, *dposv* and *sposv*. It also exists with a native C interface as *LAPACKE_zposv*.

4.6.170 zposvx

zposvx uses the Cholesky factorization A = U^H *U or A = L*L^H to compute the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N Hermitian positive definite matrix and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed:
```

1. If FACT = 'E', real scaling factors are computed to equilibrate

```
 the system:
   diag(S) * A * diag(S) * inv(diag(S)) * X = diag(S) * B
Whether or not the system will be equilibrated depends on the
scaling of the matrix A, but if equilibration is used, A is
overwritten by diag(S)*A*diag(S) and B by diag(S)*B.
```

2. If FACT = 'N' or 'E', the Cholesky decomposition is used to

```
 factor the matrix A (after equilibration if FACT = 'E') as
   A = U**H* U,  if UPLO = 'U', or
   A = L * L**H,  if UPLO = 'L',
where U is an upper triangular matrix and L is a lower triangular
matrix.
```

3. If the leading i-by-i principal minor is not positive definite,

```
 then the routine returns with INFO = i. Otherwise, the factored
form of A is used to estimate the condition number of the matrix
A.  If the reciprocal of the condition number is less than machine
precision, INFO = N+1 is returned as a warning, but the routine
still goes on to solve for X and compute error bounds as
described below.
```

4. The system of equations is solved for X using the factored form

```
of A.
```

5. Iterative refinement is applied to improve the computed solution

```
 matrix and calculate error bounds and backward error estimates
for it.
```

6. If equilibration was used, the matrix X is premultiplied by

```
 diag(S) so that it solves the original system before
equilibration.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zposvx(FACT, UPLO, N, NRHS, A, LDA, AF, LDAF, EQUED, S, B, LDB, X,
                  LDX, RCOND, FERR, BERR, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zposvx_(const char *fact, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *nrhs, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_doublecomplex_t *af,
             const armpl_int_t *ldaf, char *equed, double *s,
             armpl_doublecomplex_t *b, const armpl_int_t *ldb,
             armpl_doublecomplex_t *x, const armpl_int_t *ldx, double *rcond,
             double *ferr, double *berr, armpl_doublecomplex_t *work,
             double *rwork, armpl_int_t *info, ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AF contains the factored form of A. If EQUED = 'Y', the matrix A has been equilibrated with scaling factors given by S. A and AF will not be modified. =

'N': The matrix A will be copied to AF and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AF and factored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A, except if FACT = 'F' and EQUED = 'Y', then A must contain the equilibrated matrix diag(S)*A*diag(S). If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. A is not modified if FACT = 'F' or 'N', or if FACT = 'E' and EQUED = 'N' on exit.

On exit, if FACT = 'E' and EQUED = 'Y', A is overwritten by diag(S)*A*diag(S).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input and output parameter.

AF is COMPLEX*16

AF is an array, dimension (LDAF, N). If FACT = 'F', then AF is an input argument and on entry contains the triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H , in the same storage format as A. If EQUED .ne. 'N', then AF is the factored form of the equilibrated matrix diag(S)*A*diag(S).

If FACT = 'N', then AF is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H of the original matrix A.

If FACT = 'E', then AF is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H of the equilibrated matrix A (see the description of A for the form of the equilibrated matrix).

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'Y': Equilibration was done, i.e., A has been replaced by diag(S) * A * diag(S). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

S Input and output parameter.

S is DOUBLE PRECISION

S is an array, dimension (N). The scale factors for A; not accessed if EQUED = 'N'. S is an input argument if FACT = 'F'; otherwise, S is an output argument. If FACT = 'F' and EQUED = 'Y', each element of S must be positive.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS righthand side matrix B. On exit, if EQUED = 'N', B is not modified; if EQUED = 'Y', B is overwritten by diag(S) * B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X to the original system of equations. Note that if EQUED = 'Y', A and B are modified on exit, and the solution to the equilibrated system is inv(diag(S))*X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The estimate of the reciprocal condition number of the matrix A after equilibration (if done). If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed. RCOND = 0 is returned. = N+1: U is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *cposvx*, *dposvx* and *sposvx*. It also exists with a native C interface as *LAPACKE_zposvx*.

4.6.171 zposvxx

```
ZPOSVXX uses the Cholesky factorization A = U**T*U or A = L*L**T
   to compute the solution to a complex*16 system of linear equations
   A * X = B, where A is an N-by-N symmetric positive definite matrix
   and X and B are N-by-NRHS matrices.

   If requested, both normwise and maximum componentwise error bounds
   are returned. ZPOSVXX will return a solution with a tiny
   guaranteed error (O(eps) where eps is the working machine
   precision) unless the matrix is very ill-conditioned, in which
   case a warning is returned. Relevant condition numbers also are
   calculated and returned.

   ZPOSVXX accepts user-provided factorizations and equilibration
   factors; see the definitions of the FACT and EQUED options.
   Solving with refinement and using a factorization from a previous
   ZPOSVXX call will also produce a solution with either O(eps)
   errors or warnings, but we cannot make that claim for general
   user-provided factorizations and equilibration factors if they
   differ from what ZPOSVXX would itself produce.

                                                                  The␣
↪following steps are performed:

   1. If FACT = 'E', double precision scaling factors are computed to equilibrate
   the system:

     diag(S)*A*diag(S)      *inv(diag(S))*X = diag(S)*B

   Whether or not the system will be equilibrated depends on the
   scaling of the matrix A, but if equilibration is used, A is
   overwritten by diag(S)*A*diag(S) and B by diag(S)*B.

   2. If FACT = 'N' or 'E', the Cholesky decomposition is used to
   factor the matrix A (after equilibration if FACT = 'E') as
      A = U**T* U,  if UPLO = 'U', or
      A = L * L**T,  if UPLO = 'L',
   where U is an upper triangular matrix and L is a lower triangular
   matrix.

   3. If the leading i-by-i principal minor is not positive definite,
   then the routine returns with INFO = i. Otherwise, the factored
   form of A is used to estimate the condition number of the matrix
   A (see argument RCOND).  If the reciprocal of the condition number
   is less than machine precision, the routine still goes on to solve
   for X and compute error bounds as described below.
```

(continues on next page)

(continued from previous page)

```
   4. The system of equations is solved for X using the factored form
   of A.

   5. By default (unless PARAMS(LA_LINRX_ITREF_I) is set to zero),
   the routine will use iterative refinement to try to get a small
   error and error bounds.  Refinement calculates the residual to at
   least twice the working precision.

   6. If equilibration was used, the matrix X is premultiplied by
   diag(S) so that it solves the original system before
   equilibration.                          Some optional parameters are bundled␣
↪in the PARAMS array.  These
    settings determine how refinement is performed, but often the
    defaults are acceptable.  If the defaults are acceptable, users
    can pass NPARAMS = 0 which prevents the source code from accessing
    the PARAMS argument.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zposvxx(FACT, UPLO, N, NRHS, A, LDA, AF, LDAF, EQUED, S, B, LDB, X,
                   LDX, RCOND, RPVGRW, BERR, N_ERR_BNDS, ERR_BNDS_NORM,
                   ERR_BNDS_COMP, NPARAMS, PARAMS, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zposvxx_(const char *fact, const char *uplo, const armpl_int_t *n,
              const armpl_int_t *nrhs, armpl_doublecomplex_t *a,
              const armpl_int_t *lda, armpl_doublecomplex_t *af,
              const armpl_int_t *ldaf, char *equed, double *s,
              armpl_doublecomplex_t *b, const armpl_int_t *ldb,
              armpl_doublecomplex_t *x, const armpl_int_t *ldx, double *rcond,
              double *rpvgrw, double *berr, const armpl_int_t *n_err_bnds,
              double *err_bnds_norm, double *err_bnds_comp,
              const armpl_int_t *nparams, double *params,
              armpl_doublecomplex_t *work, double *rwork, armpl_int_t *info,
              ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AF contains the factored form of A. If EQUED is not 'N', the matrix A has been equilibrated with scaling factors given by S. A and AF are not modified. = 'N': The matrix A will be copied to AF and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AF and factored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the symmetric matrix A, except if FACT = 'F' and EQUED = 'Y', then A must contain the equilibrated matrix diag(S)*A*diag(S). If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. A is not modified if FACT = 'F' or 'N', or if FACT = 'E' and EQUED = 'N' on exit.

On exit, if FACT = 'E' and EQUED = 'Y', A is overwritten by diag(S)*A*diag(S).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input and output parameter.

AF is COMPLEX*16

AF is an array, dimension (LDAF, N). If FACT = 'F', then AF is an input argument and on entry contains the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , in the same storage format as A. If EQUED .ne. 'N', then AF is the factored form of the equilibrated matrix diag(S)*A*diag(S).

If FACT = 'N', then AF is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T of the original matrix A.

If FACT = 'E', then AF is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T of the equilibrated matrix A (see the description of A for the form of the equilibrated matrix).

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'Y': Both row and column equilibration, i.e., A has been replaced by diag(S) * A * diag(S). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

S Input and output parameter.

S is DOUBLE PRECISION

S is an array, dimension (N). The row scale factors for A. If EQUED = 'Y', A is multiplied on the left and right by diag(S). S is an input argument if FACT = 'F'; otherwise, S is an output argument. If FACT = 'F' and EQUED = 'Y', each element of S must be positive. If S is output, each element of S is a power of the radix. If S is input, each element of S should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or

overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if EQUED = 'N', B is not modified; if EQUED = 'Y', B is overwritten by diag(S)*B;

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). If INFO = 0, the N-by-NRHS solution matrix X to the original system of equations. Note that A and B are modified on exit if EQUED .ne. 'N', and the solution to the equilibrated system is inv(diag(S))*X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

RPVGRW Output parameter.

RPVGRW is DOUBLE PRECISION

Reciprocal pivot growth. On exit, this contains the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If this is much less than 1, then the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, estimated condition numbers, and error bounds could be unreliable. If factorization fails with 0<INFO<=N, then this contains the reciprocal pivot growth factor for the leading INFO columns of A.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). Componentwise relative backward error. This is the componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

N_ERR_BNDS Input parameter.

N_ERR_BNDS is INTEGER

Number of error bounds to return for each right hand side and each type (normwise or componentwise). See ERR_BNDS_NORM and ERR_BNDS_COMP below.

ERR_BNDS_NORM Output parameter.

ERR_BNDS_NORM is DOUBLE PRECISION

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * dlamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * dlamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * dlamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Output parameter.

ERR_BNDS_COMP is DOUBLE PRECISION

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:, N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * dlamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * dlamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * dlamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

NPARAMS Input parameter.

NPARAMS is INTEGER

Specifies the number of parameters set in PARAMS. If .LE. 0, the PARAMS array is never referenced and default values are used.

PARAMS Input and output parameter.

PARAMS is DOUBLE PRECISION

PARAMS is an array, dimension NPARAMS. Specifies algorithm parameters. If an entry is .LT. 0.0, then that entry will be filled with default value used for that parameter. Only positions up to NPARAMS are accessed; defaults are used for higher-numbered parameters.

PARAMS(LA_LINRX_ITREF_I = 1) : Whether to perform iterative refinement or not. Default: 1.0D+0 = 0.0 : No refinement is performed, and no error bounds are computed. = 1.0 : Use the extra-precise refinement algorithm. (other values are reserved for future use)

PARAMS(LA_LINRX_ITHRESH_I = 2) : Maximum number of residual computations allowed for refinement. Default: 10 Aggressive: Set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in err_bnds_norm and err_bnds_comp may no longer be trustworthy.

PARAMS(LA_LINRX_CWISE_I = 3) : Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Default: 1.0 (attempt componentwise convergence)

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. The solution to every right-hand side is guaranteed. < 0: If INFO = -i, the i-th argument had an illegal value > 0 and <= N: U(INFO, INFO) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+J: The solution corresponding to the Jth right-hand side is not guaranteed. The solutions corresponding to other right- hand sides K with K > J may not be guaranteed as well, but only the first such right-hand side is reported. If a small componentwise error is not requested (PARAMS(3) = 0.0) then the Jth right-hand side is the first with a normwise error bound that is not guaranteed (the smallest J such that ERR_BNDS_NORM(J,1) = 0.0). By default (PARAMS(3) = 1.0) the Jth right-hand side is the first with either a normwise or componentwise error bound that is not guaranteed (the smallest J such that either ERR_BNDS_NORM(J,1) = 0.0 or ERR_BNDS_COMP(J,1) = 0.0). See the definition of ERR_BNDS_NORM(:,1) and ERR_BNDS_COMP(:,1). To get information about all of the right-hand sides check ERR_BNDS_NORM or ERR_BNDS_COMP.

Related Information

For this routine in other precisions, please see *cposvxx*, *dposvxx* and *sposvxx*. It also exists with a native C interface as *LAPACKE_zposvxx*.

4.6.172 zpotrs

zpotrs solves a system of linear equations A*X = B with a Hermitian positive definite matrix A using the Cholesky factorization $A = U^H * U$ or $A = L * L^H$ computed by ZPOTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine zpotrs(UPLO, N, NRHS, A, LDA, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void zpotrs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_doublecomplex_t *b, const armpl_int_t *ldb,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The triangular factor U or L from the Cholesky factorization A = U^H * U or A = L * L^H , as computed by ZPOTRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cpotrs*, *dpotrs* and *spotrs*. It also exists with a native C interface as *LAPACKE_zpotrs*.

4.6.173 zppsv

zppsv computes the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N Hermitian positive definite matrix stored in packed format and X and B are N-by-NRHS matrices.

The Cholesky decomposition is used to factor A as

```
A = U**H * U,  if UPLO = 'U', or
A = L * L**H,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is a lower triangular matrix. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine zppsv(UPLO, N, NRHS, AP, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void zppsv_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
            armpl_doublecomplex_t *ap, armpl_doublecomplex_t *b,
            const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AP Input and output parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the Hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n. See below for further details.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H , in the same storage format as A.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

Related Information

For this routine in other precisions, please see *cppsv*, *dppsv* and *sppsv*. It also exists with a native C interface as *LAPACKE_zppsv*.

4.6.174 zppsvx

zppsvx uses the Cholesky factorization A = U^H * U or A = L * L^H to compute the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N Hermitian positive definite matrix stored in packed format and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed:
```

1. If FACT = 'E', real scaling factors are computed to equilibrate

```
 the system:
   diag(S) * A * diag(S) * inv(diag(S)) * X = diag(S) * B
Whether or not the system will be equilibrated depends on the
scaling of the matrix A, but if equilibration is used, A is
overwritten by diag(S)*A*diag(S) and B by diag(S)*B.
```

2. If FACT = 'N' or 'E', the Cholesky decomposition is used to

```
 factor the matrix A (after equilibration if FACT = 'E') as
   A = U**H * U ,  if UPLO = 'U', or
   A = L * L**H,  if UPLO = 'L',
where U is an upper triangular matrix, L is a lower triangular
matrix, and **H indicates conjugate transpose.
```

3. If the leading i-by-i principal minor is not positive definite,

```
 then the routine returns with INFO = i. Otherwise, the factored
form of A is used to estimate the condition number of the matrix
A.  If the reciprocal of the condition number is less than machine
precision, INFO = N+1 is returned as a warning, but the routine
```

(continues on next page)

(continued from previous page)

```
still goes on to solve for X and compute error bounds as
described below.
```

4. The system of equations is solved for X using the factored form

```
of A.
```

5. Iterative refinement is applied to improve the computed solution

```
 matrix and calculate error bounds and backward error estimates
for it.
```

6. If equilibration was used, the matrix X is premultiplied by

```
 diag(S) so that it solves the original system before
equilibration.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zppsvx(FACT, UPLO, N, NRHS, AP, AFP, EQUED, S, B, LDB, X, LDX,
                  RCOND, FERR, BERR, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zppsvx_(const char *fact, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *nrhs, armpl_doublecomplex_t *ap,
             armpl_doublecomplex_t *afp, char *equed, double *s,
             armpl_doublecomplex_t *b, const armpl_int_t *ldb,
             armpl_doublecomplex_t *x, const armpl_int_t *ldx, double *rcond,
             double *ferr, double *berr, armpl_doublecomplex_t *work,
             double *rwork, armpl_int_t *info, ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AFP contains the factored form of A. If EQUED = 'Y', the matrix A has been equilibrated with scaling factors given by S. AP and AFP will not be modified. = 'N': The matrix A will be copied to AFP and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AFP and factored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AP Input and output parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the Hermitian matrix A, packed columnwise in a linear array, except if FACT = 'F' and EQUED = 'Y', then A must contain the equilibrated matrix diag(S)*A*diag(S). The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n. See below for further details. A is not modified if FACT = 'F' or 'N', or if FACT = 'E' and EQUED = 'N' on exit.

On exit, if FACT = 'E' and EQUED = 'Y', A is overwritten by diag(S)*A*diag(S).

AFP Input and output parameter.

AFP is COMPLEX*16

AFP is an array, dimension (N*(N+1)/2). If FACT = 'F', then AFP is an input argument and on entry contains the triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H , in the same storage format as A. If EQUED .ne. 'N', then AFP is the factored form of the equilibrated matrix A.

If FACT = 'N', then AFP is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization A = U^H * U or A = L * L^H of the original matrix A.

If FACT = 'E', then AFP is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization A = U^H * U or A = L * L^H of the equilibrated matrix A (see the description of AP for the form of the equilibrated matrix).

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'Y': Equilibration was done, i.e., A has been replaced by diag(S) * A * diag(S). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

S Input and output parameter.

S is DOUBLE PRECISION

S is an array, dimension (N). The scale factors for A; not accessed if EQUED = 'N'. S is an input argument if FACT = 'F'; otherwise, S is an output argument. If FACT = 'F' and EQUED = 'Y', each element of S must be positive.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if EQUED = 'N', B is not modified; if EQUED = 'Y', B is overwritten by diag(S) * B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X to the original system of equations. Note that if EQUED = 'Y', A and B are modified on exit, and the solution to the equilibrated system is inv(diag(S))*X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The estimate of the reciprocal condition number of the matrix A after equilibration (if done). If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed. RCOND = 0 is returned. = N+1: U is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *cppsvx*, *dppsvx* and *sppsvx*. It also exists with a native C interface as *LAPACKE_zppsvx*.

4.6.175 zpptrs

zpptrs solves a system of linear equations A*X = B with a Hermitian positive definite matrix A in packed storage using the Cholesky factorization $A = U^H * U$ or $A = L * L^H$ computed by ZPPTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine zpptrs(UPLO, N, NRHS, AP, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void zpptrs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_doublecomplex_t *ap, armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AP Input parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). The triangular factor U or L from the Cholesky factorization A = U^H * U or A = L * L^H , packed columnwise in a linear array. The j-th column of U or L is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = U(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = L(i,j) for j<=i<=n.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cpptrs*, *dpptrs* and *spptrs*. It also exists with a native C interface as *LAPACKE_zpptrs*.

4.6.176 zptsv

`zptsv` computes the solution to a complex system of linear equations A*X = B, where A is an N-by-N Hermitian positive definite tridiagonal matrix, and X and B are N-by-NRHS matrices.

A is factored as A = L*D*L^H , and the factored form of A is then used to solve the system of equations.

Syntax

Fortran specification:

```
use armpl_library

subroutine zptsv(N, NRHS, D, E, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void zptsv_(const armpl_int_t *n, const armpl_int_t *nrhs, double *d,
            armpl_doublecomplex_t *e, armpl_doublecomplex_t *b,
            const armpl_int_t *ldb, armpl_int_t *info);
```

Parameters

N Input parameter.

`N` is `INTEGER`

The order of the matrix A. `N` >= 0.

NRHS Input parameter.

`NRHS` is `INTEGER`

The number of right hand sides, i.e., the number of columns of the matrix `B`. `NRHS` >= 0.

D Input and output parameter.

`D` is `DOUBLE PRECISION`

`D` is an array, dimension (`N`). On entry, the n diagonal elements of the tridiagonal matrix A. On exit, the n diagonal elements of the diagonal matrix `D` from the factorization A = L*D*L^H .

E Input and output parameter.

`E` is `COMPLEX*16`

`E` is an array, dimension (N-1). On entry, the (n-1) subdiagonal elements of the tridiagonal matrix A. On exit, the (n-1) subdiagonal elements of the unit bidiagonal factor L from the L*D*L^H factorization of A. `E` can also be regarded as the superdiagonal of the unit bidiagonal factor U from the U^H *D*U factorization of A.

B Input and output parameter.

`B` is `COMPLEX*16`

`B` is an array, dimension (`LDB`, `NRHS`). On entry, the N-by-NRHS right hand side matrix `B`. On exit, if `INFO` = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

`LDB` is `INTEGER`

The leading dimension of the array `B`. `LDB` >= max(1, `N`).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the leading minor of order i is not positive definite, and the solution has not been computed. The factorization has not been completed unless i = N.

Related Information

For this routine in other precisions, please see *cptsv*, *dptsv* and *sptsv*. It also exists with a native C interface as *LAPACKE_zptsv*.

4.6.177 zptsvx

zptsvx uses the factorization A = L*D*L^H to compute the solution to a complex system of linear equations A*X = B, where A is an N-by-N Hermitian positive definite tridiagonal matrix and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed:
```

1. If FACT = 'N', the matrix A is factored as A = L*D*L^H , where L

```
 is a unit lower bidiagonal matrix and D is diagonal.  The
factorization can also be regarded as having the form
A = U**H*D*U.
```

2. If the leading i-by-i principal minor is not positive definite,

```
 then the routine returns with INFO = i. Otherwise, the factored
form of A is used to estimate the condition number of the matrix
A.  If the reciprocal of the condition number is less than machine
precision, INFO = N+1 is returned as a warning, but the routine
still goes on to solve for X and compute error bounds as
described below.
```

3. The system of equations is solved for X using the factored form

```
of A.
```

4. Iterative refinement is applied to improve the computed solution

```
 matrix and calculate error bounds and backward error estimates
for it.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zptsvx(FACT, N, NRHS, D, E, DF, EF, B, LDB, X, LDX, RCOND, FERR,
                  BERR, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zptsvx_(const char *fact, const armpl_int_t *n, const armpl_int_t *nrhs,
             const double *d, const armpl_doublecomplex_t *e, double *df,
             armpl_doublecomplex_t *ef, const armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, armpl_doublecomplex_t *x,
             const armpl_int_t *ldx, double *rcond, double *ferr,
             double *berr, armpl_doublecomplex_t *work, double *rwork,
             armpl_int_t *info, ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry. = 'F': On entry, DF and EF contain the factored form of A. D, E, DF, and EF will not be modified. = 'N': The matrix A will be copied to DF and EF and factored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The n diagonal elements of the tridiagonal matrix A.

E Input parameter.

E is COMPLEX*16

E is an array, dimension (N-1). The (n-1) subdiagonal elements of the tridiagonal matrix A.

DF Input and output parameter.

DF is DOUBLE PRECISION

DF is an array, dimension (N). If FACT = 'F', then DF is an input argument and on entry contains the n diagonal elements of the diagonal matrix D from the L*D*L^H factorization of A. If FACT = 'N', then DF is an output argument and on exit contains the n diagonal elements of the diagonal matrix D from the L*D*L^H factorization of A.

EF Input and output parameter.

EF is COMPLEX*16

EF is an array, dimension (N-1). If FACT = 'F', then EF is an input argument and on entry contains the (n-1) subdiagonal elements of the unit bidiagonal factor L from the L*D*L^H factorization of A. If FACT = 'N', then EF is an output argument and on exit contains the (n-1) subdiagonal elements of the unit bidiagonal factor L from the L*D*L^H factorization of A.

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). The N-by-NRHS right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The reciprocal condition number of the matrix A. If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j).

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed. RCOND = 0 is returned. = N+1: U is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *cptsvx*, *dptsvx* and *sptsvx*. It also exists with a native C interface as *LAPACKE_zptsvx*.

4.6.178 zpttrs

zpttrs solves a tridiagonal system of the form

```
A * X = B
```

using the factorization A = U^H *D* U or A = L*D*L^H computed by ZPTTRF. D is a diagonal matrix specified in the vector D, U (or L) is a unit bidiagonal matrix whose superdiagonal (subdiagonal) is specified in the vector E, and X and B are N by NRHS matrices.

Syntax

Fortran specification:

```
use armpl_library

subroutine zpttrs(UPLO, N, NRHS, D, E, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void zpttrs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const double *d, const armpl_doublecomplex_t *e,
             armpl_doublecomplex_t *b, const armpl_int_t *ldb,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies the form of the factorization and whether the vector E is the superdiagonal of the upper bidiagonal factor U or the subdiagonal of the lower bidiagonal factor L. = 'U': A = U^H *D*U, E is the superdiagonal of U = 'L': A = L*D*L^H , E is the subdiagonal of L

N Input parameter.

N is INTEGER

The order of the tridiagonal matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The n diagonal elements of the diagonal matrix D from the factorization A = U^H *D*U or A = L*D*L^H .

E Input parameter.

E is COMPLEX*16

E is an array, dimension (N-1). If UPLO = 'U', the (n-1) superdiagonal elements of the unit bidiagonal factor U from the factorization A = U^H *D*U. If UPLO = 'L', the (n-1) subdiagonal elements of the unit bidiagonal factor L from the factorization A = L*D*L^H .

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the right hand side vectors B for the system of linear equations. On exit, the solution vectors, X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cpttrs*, *dpttrs* and *spttrs*. It also exists with a native C interface as *LAPACKE_zpttrs*.

4.6.179 zspsv

zspsv computes the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N symmetric matrix stored in packed format and X and B are N-by-NRHS matrices.

The diagonal pivoting method is used to factor A as

```
A = U * D * U**T,  if UPLO = 'U', or
A = L * D * L**T,  if UPLO = 'L',
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine zspsv(UPLO, N, NRHS, AP, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void zspsv_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
            armpl_doublecomplex_t *ap, armpl_int_t *ipiv,
            armpl_doublecomplex_t *b, const armpl_int_t *ldb,
            armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AP Input and output parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n. See below for further details.

On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by ZSPTRF, stored as a packed triangular matrix in the same storage format as A.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D, as determined by ZSPTRF. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged, and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, so the solution could not be computed.

Related Information

For this routine in other precisions, please see *cspsv*, *dspsv* and *sspsv*. It also exists with a native C interface as *LAPACKE_zspsv*.

4.6.180 zspsvx

zspsvx uses the diagonal pivoting factorization A = U*D*U^T or A = L*D*L^T to compute the solution to a complex system of linear equations A * X = B, where A is an N-by-N symmetric matrix stored in packed format and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed:
```

1. If FACT = 'N', the diagonal pivoting method is used to factor A as

```
   A = U * D * U**T,  if UPLO = 'U', or
   A = L * D * L**T,  if UPLO = 'L',
where U (or L) is a product of permutation and unit upper (lower)
triangular matrices and D is symmetric and block diagonal with
1-by-1 and 2-by-2 diagonal blocks.
```

2. If some D(i,i)=0, so that D is exactly singular, then the routine

```
 returns with INFO = i. Otherwise, the factored form of A is used
to estimate the condition number of the matrix A.  If the
reciprocal of the condition number is less than machine precision,
INFO = N+1 is returned as a warning, but the routine still goes on
to solve for X and compute error bounds as described below.
```

3. The system of equations is solved for X using the factored form

```
of A.
```

4. Iterative refinement is applied to improve the computed solution

```
 matrix and calculate error bounds and backward error estimates
for it.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zspsvx(FACT, UPLO, N, NRHS, AP, AFP, IPIV, B, LDB, X, LDX, RCOND,
                  FERR, BERR, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zspsvx_(const char *fact, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *nrhs, const armpl_doublecomplex_t *ap,
             armpl_doublecomplex_t *afp, armpl_int_t *ipiv,
             const armpl_doublecomplex_t *b, const armpl_int_t *ldb,
             armpl_doublecomplex_t *x, const armpl_int_t *ldx, double *rcond,
             double *ferr, double *berr, armpl_doublecomplex_t *work,
             double *rwork, armpl_int_t *info, ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of A has been supplied on entry. = 'F': On entry, AFP and IPIV contain the factored form of A. AP, AFP and IPIV will not be modified. = 'N': The matrix A will be copied to AFP and factored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AP Input parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). The upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n. See below for further details.

AFP Input and output parameter.

AFP is COMPLEX*16

AFP is an array, dimension (N*(N+1)/2). If FACT = 'F', then AFP is an input argument and on entry contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by ZSPTRF, stored as a packed triangular matrix in the same storage format as A.

If FACT = 'N', then AFP is an output argument and on exit contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by ZSPTRF, stored as a packed triangular matrix in the same storage format as A.

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains details of the interchanges and the block structure of D, as determined by ZSPTRF. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

If FACT = 'N', then IPIV is an output argument and on exit contains details of the interchanges and the block structure of D, as determined by ZSPTRF.

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). The N-by-NRHS right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The estimate of the reciprocal condition number of the matrix A. If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: D(i,i) is exactly zero. The factorization has been completed but the factor D is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+1: D is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *cspsvx*, *dspsvx* and *sspsvx*. It also exists with a native C interface as *LAPACKE_zspsvx*.

4.6.181 zsptrs

zsptrs solves a system of linear equations A*X = B with a complex symmetric matrix A stored in packed format using the factorization A = U*D*U^T or A = L*D*L^T computed by ZSPTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsptrs(UPLO, N, NRHS, AP, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void zsptrs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_doublecomplex_t *ap, const armpl_int_t *ipiv,
             armpl_doublecomplex_t *b, const armpl_int_t *ldb,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AP Input parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by ZSPTRF, stored as a packed triangular matrix.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZSPTRF.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csptrs*, *dsptrs* and *ssptrs*. It also exists with a native C interface as *LAPACKE_zsptrs*.

4.6.182 zsysv

zsysv computes the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N symmetric matrix and X and B are N-by-NRHS matrices.

The diagonal pivoting method is used to factor A as

```
A = U * D * U**T,  if UPLO = 'U', or
A = L * D * L**T,  if UPLO = 'L',
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsysv(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zsysv_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
            armpl_doublecomplex_t *a, const armpl_int_t *lda,
            armpl_int_t *ipiv, armpl_doublecomplex_t *b,
            const armpl_int_t *ldb, armpl_doublecomplex_t *work,
            const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by ZSYTRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D, as determined by ZSYTRF. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged, and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= 1, and for best performance LWORK >= max(1,N*NB), where NB is the optimal blocksize for ZSYTRF. for LWORK < N, TRS will be done with Level BLAS 2 for LWORK >= N, TRS will be done with Level BLAS 3

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, so the solution could not be computed.

Related Information

For this routine in other precisions, please see *csysv*, *dsysv* and *ssysv*. It also exists with a native C interface as *LAPACKE_zsysv*.

4.6.183 zsysv_aa

ZSYSV computes the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N symmetric matrix and X and B are N-by-NRHS matrices.

Aasen's algorithm is used to factor A as

```
A = U * T * U**T,  if UPLO = 'U', or
A = L * T * L**T,  if UPLO = 'L',
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and T is symmetric tridiagonal. The factored form of A is then used to solve the system of equations A * X = B.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsysv_aa(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zsysv_aa_(const char *uplo, const armpl_int_t *n,
               const armpl_int_t *nrhs, armpl_doublecomplex_t *a,
               const armpl_int_t *lda, armpl_int_t *ipiv,
               armpl_doublecomplex_t *b, const armpl_int_t *ldb,
               armpl_doublecomplex_t *work, const armpl_int_t *lwork,
               armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the tridiagonal matrix T and the multipliers used to obtain the factor U or L from the factorization A = U*T*U^T or A = L*T*L^T as computed by ZSYTRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

On exit, it contains the details of the interchanges, i.e., the row and column k of A were interchanged with the row and column IPIV(k).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= MAX(1,2*N,3*N-2), and for the best performance, LWORK >= MAX(1,N*NB), where NB is the optimal blocksize for ZSYTRF_AA.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, so the solution could not be computed.

Related Information

For this routine in other precisions, please see *csysv_aa*, *dsysv_aa* and *ssysv_aa*. It also exists with a native C interface as *LAPACKE_zsysv_aa*.

4.6.184 zsysv_rk

ZSYSV_RK computes the solution to a complex system of linear equations A * X = B, where A is an N-by-N symmetric matrix and X and B are N-by-NRHS matrices.

The bounded Bunch-Kaufman (rook) diagonal pivoting method is used to factor A as

```
A = P*U*D*(U**T)*(P**T),  if UPLO = 'U', or
A = P*L*D*(L**T)*(P**T),  if UPLO = 'L',
```

where U (or L) is unit upper (or lower) triangular matrix, U^T (or L^T) is the transpose of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

ZSYTRF_RK is called to compute the factorization of a complex symmetric matrix. The factored form of A is then used to solve the system of equations A * X = B by calling BLAS3 routine ZSYTRS_3.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsysv_rk(UPLO, N, NRHS, A, LDA, E, IPIV, B, LDB, WORK, LWORK,
                    INFO)
```

C specification:

```
#include "armpl.h"

void zsysv_rk_(const char *uplo, const armpl_int_t *n,
               const armpl_int_t *nrhs, armpl_doublecomplex_t *a,
               const armpl_int_t *lda, armpl_doublecomplex_t *e,
               armpl_int_t *ipiv, armpl_doublecomplex_t *b,
               const armpl_int_t *ldb, armpl_doublecomplex_t *work,
               const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U': the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced.

If UPLO = 'L': the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, diagonal of the block diagonal matrix D and factors U or L as computed by ZSYTRF_RK: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

For more info see the description of ZSYTRF_RK routine.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Output parameter.

E is COMPLEX*16

E is an array, dimension (N). On exit, contains the output computed by the factorization routine ZSYTRF_RK, i.e. the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) is set to 0; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) is set to 0.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is set to 0 in both UPLO = 'U' or UPLO = 'L' cases.

For more info see the description of ZSYTRF_RK routine.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D, as determined by ZSYTRF_RK.

For more info see the description of ZSYTRF_RK routine.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)).. Work array used in the factorization stage. On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= 1. For best performance of factorization stage LWORK >= max(1,N*NB), where NB is the optimal blocksize for ZSYTRF_RK.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array for factorization stage, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit

< 0: If INFO = -k, the k-th argument had an illegal value

> 0: If INFO = k, the matrix A is singular, because: If UPLO = 'U': column k in the upper triangular part of A contains all zeros. If UPLO = 'L': column k in the lower triangular part of A contains all zeros.

Therefore D(k,k) is exactly zero, and superdiagonal elements of column k of U (or subdiagonal elements of column k of L) are all zeros. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

NOTE: INFO only stores the first occurrence of a singularity, any subsequent occurrence of singularity is not stored in INFO even though the factorization always completes.

Related Information

For this routine in other precisions, please see *csysv_rk*, *dsysv_rk* and *ssysv_rk*. It also exists with a native C interface as *LAPACKE_zsysv_rk*.

4.6.185 zsysv_rook

ZSYSV_ROOK computes the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N symmetric matrix and X and B are N-by-NRHS matrices.

The diagonal pivoting method is used to factor A as

```
A = U * D * U**T,  if UPLO = 'U', or
A = L * D * L**T,  if UPLO = 'L',
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

ZSYTRF_ROOK is called to compute the factorization of a complex symmetric matrix A using the bounded Bunch-Kaufman ("rook") diagonal pivoting method.

The factored form of A is then used to solve the system of equations A * X = B by calling ZSYTRS_ROOK.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsysv_rook(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zsysv_rook_(const char *uplo, const armpl_int_t *n,
                 const armpl_int_t *nrhs, armpl_doublecomplex_t *a,
```

(continues on next page)

(continued from previous page)

```
                const armpl_int_t *lda, armpl_int_t *ipiv,
                armpl_doublecomplex_t *b, const armpl_int_t *ldb,
                armpl_doublecomplex_t *work, const armpl_int_t *lwork,
                armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by ZSYTRF_ROOK.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D, as determined by ZSYTRF_ROOK.

If UPLO = 'U': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k-1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k-1 and -IPIV(k-1) were inerchaged, D(k-1:k,k-1:k) is a 2-by-2 diagonal block.

If UPLO = 'L': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k+1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k+1 and -IPIV(k+1) were inerchaged, D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= 1, and for best performance LWORK >= max(1,N*NB), where NB is the optimal blocksize for ZSYTRF_ROOK.

TRS will be done with Level 2 BLAS

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, so the solution could not be computed.

Related Information

For this routine in other precisions, please see *csysv_rook*, *dsysv_rook* and *ssysv_rook*. It also exists with a native C interface as *LAPACKE_zsysv_rook*.

4.6.186 zsysvx

zsysvx uses the diagonal pivoting factorization to compute the solution to a complex system of linear equations A * X = B, where A is an N-by-N symmetric matrix and X and B are N-by-NRHS matrices.

Error bounds on the solution and a condition estimate are also provided.

```
The following steps are performed:
```

1. If FACT = 'N', the diagonal pivoting method is used to factor A.

```
 The form of the factorization is
   A = U * D * U**T,  if UPLO = 'U', or
   A = L * D * L**T,  if UPLO = 'L',
where U (or L) is a product of permutation and unit upper (lower)
triangular matrices, and D is symmetric and block diagonal with
1-by-1 and 2-by-2 diagonal blocks.
```

2. If some D(i,i)=0, so that D is exactly singular, then the routine

```
 returns with INFO = i. Otherwise, the factored form of A is used
to estimate the condition number of the matrix A.  If the
reciprocal of the condition number is less than machine precision,
INFO = N+1 is returned as a warning, but the routine still goes on
to solve for X and compute error bounds as described below.
```

3. The system of equations is solved for X using the factored form

```
of A.
```

4. Iterative refinement is applied to improve the computed solution

```
 matrix and calculate error bounds and backward error estimates
for it.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zsysvx(FACT, UPLO, N, NRHS, A, LDA, AF, LDAF, IPIV, B, LDB, X, LDX,
                  RCOND, FERR, BERR, WORK, LWORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zsysvx_(const char *fact, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *nrhs, const armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_doublecomplex_t *af,
             const armpl_int_t *ldaf, armpl_int_t *ipiv,
             const armpl_doublecomplex_t *b, const armpl_int_t *ldb,
             armpl_doublecomplex_t *x, const armpl_int_t *ldx, double *rcond,
             double *ferr, double *berr, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, double *rwork, armpl_int_t *info,
             ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of A has been supplied on entry. = 'F': On entry, AF and IPIV contain the factored form of A. A, AF and IPIV will not be modified. = 'N': The matrix A will be copied to AF and factored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input and output parameter.

AF is COMPLEX*16

AF is an array, dimension (LDAF, N). If FACT = 'F', then AF is an input argument and on entry contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by ZSYTRF.

If FACT = 'N', then AF is an output argument and on exit returns the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T .

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains details of the interchanges and the block structure of D, as determined by ZSYTRF. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

If FACT = 'N', then IPIV is an output argument and on exit contains details of the interchanges and the block structure of D, as determined by ZSYTRF.

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). The N-by-NRHS right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The estimate of the reciprocal condition number of the matrix A. If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of WORK. LWORK >= max(1,2*N), and for best performance, when FACT = 'N', LWORK >= max(1,2*N,N*NB), where NB is the optimal blocksize for ZSYTRF.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= N: D(i,i) is exactly zero. The factorization has been completed but the factor D is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+1: D is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

Related Information

For this routine in other precisions, please see *csysvx*, *dsysvx* and *ssysvx*. It also exists with a native C interface as *LAPACKE_zsysvx*.

4.6.187 zsysvxx

```
ZSYSVXX uses the diagonal pivoting factorization to compute the
   solution to a complex*16 system of linear equations A * X = B, where
   A is an N-by-N symmetric matrix and X and B are N-by-NRHS
   matrices.

   If requested, both normwise and maximum componentwise error bounds
```

(continues on next page)

(continued from previous page)

```
   are returned. ZSYSVXX will return a solution with a tiny
   guaranteed error (O(eps) where eps is the working machine
   precision) unless the matrix is very ill-conditioned, in which
   case a warning is returned. Relevant condition numbers also are
   calculated and returned.

   ZSYSVXX accepts user-provided factorizations and equilibration
   factors; see the definitions of the FACT and EQUED options.
   Solving with refinement and using a factorization from a previous
   ZSYSVXX call will also produce a solution with either O(eps)
   errors or warnings, but we cannot make that claim for general
   user-provided factorizations and equilibration factors if they
   differ from what ZSYSVXX would itself produce.

                                                                  The␣
↪following steps are performed:

   1. If FACT = 'E', double precision scaling factors are computed to equilibrate
   the system:

     diag(S)*A*diag(S)      *inv(diag(S))*X = diag(S)*B

   Whether or not the system will be equilibrated depends on the
   scaling of the matrix A, but if equilibration is used, A is
   overwritten by diag(S)*A*diag(S) and B by diag(S)*B.

   2. If FACT = 'N' or 'E', the LU decomposition is used to factor
   the matrix A (after equilibration if FACT = 'E') as

      A = U * D * U**T,  if UPLO = 'U', or
      A = L * D * L**T,  if UPLO = 'L',

   where U (or L) is a product of permutation and unit upper (lower)
   triangular matrices, and D is symmetric and block diagonal with
   1-by-1 and 2-by-2 diagonal blocks.

   3. If some D(i,i)=0, so that D is exactly singular, then the
   routine returns with INFO = i. Otherwise, the factored form of A
   is used to estimate the condition number of the matrix A (see
   argument RCOND).  If the reciprocal of the condition number is
   less than machine precision, the routine still goes on to solve
   for X and compute error bounds as described below.

   4. The system of equations is solved for X using the factored form
   of A.

   5. By default (unless PARAMS(LA_LINRX_ITREF_I) is set to zero),
   the routine will use iterative refinement to try to get a small
   error and error bounds.  Refinement calculates the residual to at
   least twice the working precision.

   6. If equilibration was used, the matrix X is premultiplied by
   diag(R) so that it solves the original system before
   equilibration.                          Some optional parameters are bundled␣
↪in the PARAMS array.  These
    settings determine how refinement is performed, but often the
    defaults are acceptable.  If the defaults are acceptable, users
    can pass NPARAMS = 0 which prevents the source code from accessing
    the PARAMS argument.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zsysvxx(FACT, UPLO, N, NRHS, A, LDA, AF, LDAF, IPIV, EQUED, S, B,
                   LDB, X, LDX, RCOND, RPVGRW, BERR, N_ERR_BNDS,
                   ERR_BNDS_NORM, ERR_BNDS_COMP, NPARAMS, PARAMS, WORK, RWORK,
                   INFO)
```

C specification:

```
#include "armpl.h"

void zsysvxx_(const char *fact, char *uplo, const armpl_int_t *n,
              const armpl_int_t *nrhs, armpl_doublecomplex_t *a,
              const armpl_int_t *lda, armpl_doublecomplex_t *af,
              const armpl_int_t *ldaf, armpl_int_t *ipiv, char *equed,
              double *s, armpl_doublecomplex_t *b, const armpl_int_t *ldb,
              armpl_doublecomplex_t *x, const armpl_int_t *ldx, double *rcond,
              double *rpvgrw, double *berr, const armpl_int_t *n_err_bnds,
              double *err_bnds_norm, double *err_bnds_comp,
              const armpl_int_t *nparams, double *params,
              armpl_doublecomplex_t *work, double *rwork, armpl_int_t *info,
              ... );
```

Parameters

FACT Input parameter.

FACT is CHARACTER*1

Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored. = 'F': On entry, AF and IPIV contain the factored form of A. If EQUED is not 'N', the matrix A has been equilibrated with scaling factors given by S. A, AF, and IPIV are not modified. = 'N': The matrix A will be copied to AF and factored. = 'E': The matrix A will be equilibrated if necessary, then copied to AF and factored.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if FACT = 'E' and EQUED = 'Y', A is overwritten by diag(S)*A*diag(S).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input and output parameter.

AF is COMPLEX*16

AF is an array, dimension (LDAF, N). If FACT = 'F', then AF is an input argument and on entry contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by DSYTRF.

If FACT = 'N', then AF is an output argument and on exit returns the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T .

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

If FACT = 'F', then IPIV is an input argument and on entry contains details of the interchanges and the block structure of D, as determined by DSYTRF. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

If FACT = 'N', then IPIV is an output argument and on exit contains details of the interchanges and the block structure of D, as determined by DSYTRF.

EQUED Input and output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration (always true if FACT = 'N'). = 'Y': Both row and column equilibration, i.e., A has been replaced by diag(S) * A * diag(S). EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

S Input and output parameter.

S is DOUBLE PRECISION

S is an array, dimension (N). The scale factors for A. If EQUED = 'Y', A is multiplied on the left and right by diag(S). S is an input argument if FACT = 'F'; otherwise, S is an output argument. If FACT = 'F' and EQUED = 'Y', each element of S must be positive. If S is output, each element of S is a power of the radix. If S is input, each element of S should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the N-by-NRHS right hand side matrix B. On exit, if EQUED = 'N', B is not modified; if EQUED = 'Y', B is overwritten by diag(S)*B;

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). If INFO = 0, the N-by-NRHS solution matrix X to the original system of equations. Note that A and B are modified on exit if EQUED .ne. 'N', and the solution to the equilibrated system is inv(diag(S))*X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

RPVGRW Output parameter.

RPVGRW is DOUBLE PRECISION

Reciprocal pivot growth. On exit, this contains the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If this is much less than 1, then the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, estimated condition numbers, and error bounds could be unreliable. If factorization fails with 0<INFO<=N, then this contains the reciprocal pivot growth factor for the leading INFO columns of A.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). Componentwise relative backward error. This is the componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

N_ERR_BNDS Input parameter.

N_ERR_BNDS is INTEGER

Number of error bounds to return for each right hand side and each type (normwise or componentwise). See ERR_BNDS_NORM and ERR_BNDS_COMP below.

ERR_BNDS_NORM Output parameter.

ERR_BNDS_NORM is DOUBLE PRECISION

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * dlamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * dlamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * dlamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Output parameter.

ERR_BNDS_COMP is DOUBLE PRECISION

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:, N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * dlamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * dlamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * dlamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

NPARAMS Input parameter.

NPARAMS is INTEGER

Specifies the number of parameters set in PARAMS. If .LE. 0, the PARAMS array is never referenced and default values are used.

PARAMS Input and output parameter.

PARAMS is DOUBLE PRECISION

PARAMS is an array, dimension NPARAMS. Specifies algorithm parameters. If an entry is .LT. 0.0, then that entry will be filled with default value used for that parameter. Only positions up to NPARAMS are accessed; defaults are used for higher-numbered parameters.

PARAMS(LA_LINRX_ITREF_I = 1) : Whether to perform iterative refinement or not. Default: 1.0D+0 = 0.0 : No refinement is performed, and no error bounds are computed. = 1.0 : Use the extra-precise refinement algorithm. (other values are reserved for future use)

PARAMS(LA_LINRX_ITHRESH_I = 2) : Maximum number of residual computations allowed for refinement. Default: 10 Aggressive: Set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in err_bnds_norm and err_bnds_comp may no longer be trustworthy.

PARAMS(LA_LINRX_CWISE_I = 3) : Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Default: 1.0 (attempt componentwise convergence)

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. The solution to every right-hand side is guaranteed. < 0: If INFO = -i, the i-th argument had an illegal value > 0 and <= N: U(INFO, INFO) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+J: The solution corresponding to the Jth right-hand side is not guaranteed. The solutions corresponding to other right- hand sides K with K > J may not be guaranteed as well, but only the first such right-hand side is reported. If a small componentwise error is not requested (PARAMS(3) = 0.0) then the Jth right-hand side is the first with a normwise error bound that is not guaranteed (the smallest J such that ERR_BNDS_NORM(J,1) = 0.0). By default (PARAMS(3) = 1.0) the Jth right-hand side is the first with either a normwise or componentwise error bound that is not guaranteed (the smallest J such that either ERR_BNDS_NORM(J,1) = 0.0 or ERR_BNDS_COMP(J,1) = 0.0). See the definition of ERR_BNDS_NORM(:,1) and ERR_BNDS_COMP(:,1). To get information about all of the right-hand sides check ERR_BNDS_NORM or ERR_BNDS_COMP.

Related Information

For this routine in other precisions, please see *csysvxx*, *dsysvxx* and *ssysvxx*. It also exists with a native C interface as *LAPACKE_zsysvxx*.

4.6.188 zsytrs

zsytrs solves a system of linear equations A*X = B with a complex symmetric matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by ZSYTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsytrs(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void zsytrs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_int_t *ipiv, armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by ZSYTRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZSYTRF.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csytrs*, *dsytrs* and *ssytrs*. It also exists with a native C interface as *LAPACKE_zsytrs*.

4.6.189 zsytrs2

zsytrs2 solves a system of linear equations A*X = B with a real symmetric matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by ZSYTRF and converted by ZSYCONV.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsytrs2(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zsytrs2_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
              const armpl_doublecomplex_t *a, const armpl_int_t *lda,
              const armpl_int_t *ipiv, armpl_doublecomplex_t *b,
              const armpl_int_t *ldb, armpl_doublecomplex_t *work,
              armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by ZSYTRF. Note that A is input / output. This might be counter-intuitive, and one may think that A is input only. A is input / output. This is because, at the start of the subroutine, we permute A in a "better" form and then we permute A back to its original form at the end.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZSYTRF.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csytrs2*, *dsytrs2* and *ssytrs2*. It also exists with a native C interface as *LAPACKE_zsytrs2*.

4.6.190 zsytrs_3

ZSYTRS_3 solves a system of linear equations A * X = B with a complex symmetric matrix A using the factorization computed by ZSYTRF_RK or ZSYTRF_BK:

```
A = P*U*D*(U**T)*(P**T) or A = P*L*D*(L**T)*(P**T),
```

where U (or L) is unit upper (or lower) triangular matrix, U^T (or L^T) is the transpose of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This algorithm is using Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsytrs_3(UPLO, N, NRHS, A, LDA, E, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void zsytrs_3_(const char *uplo, const armpl_int_t *n,
               const armpl_int_t *nrhs, const armpl_doublecomplex_t *a,
               const armpl_int_t *lda, const armpl_doublecomplex_t *e,
               const armpl_int_t *ipiv, armpl_doublecomplex_t *b,
               const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix: = 'U': Upper triangular, form is A = P*U*D*(U^T)*(P^T); = 'L': Lower triangular, form is A = P*L*D*(L^T)*(P^T).

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). Diagonal of the block diagonal matrix D and factors U or L as computed by ZSYTRF_RK and ZSYTRF_BK: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D should be provided on entry in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Input parameter.

E is COMPLEX*16

E is an array, dimension (N). On entry, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i),i=2:N, E(1) not referenced; If UPLO = 'L': E(i) = D(i+1,i),i=1:N-1, E(N) not referenced.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is not referenced in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZSYTRF_RK or ZSYTRF_BK.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csytrs_3*, *dsytrs_3* and *ssytrs_3*. It also exists with a native C interface as *LAPACKE_zsytrs_3*.

4.6.191 zsytrs_aa

ZSYTRS_AA solves a system of linear equations A*X = B with a complex symmetric matrix A using the factorization A = U*T*U^T or A = L*T*L^T computed by ZSYTRF_AA.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsytrs_aa(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zsytrs_aa_(const char *uplo, const armpl_int_t *n,
                const armpl_int_t *nrhs, const armpl_doublecomplex_t *a,
                const armpl_int_t *lda, const armpl_int_t *ipiv,
                armpl_doublecomplex_t *b, const armpl_int_t *ldb,
                const armpl_doublecomplex_t *work, const armpl_int_t *lwork,
                armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*T*U^T ; = 'L': Lower triangular, form is A = L*T*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). Details of factors computed by ZSYTRF_AA.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges as computed by ZSYTRF_AA.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Input parameter.

WORK is DOUBLE array, dimension (MAX(1, LWORK))

LWORK Input parameter.

LWORK is INTEGER, LWORK >= MAX(1,3*N-2).

param[out] INFO verbatim INFO is INTEGER = 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csytrs_aa*, *dsytrs_aa* and *ssytrs_aa*. It also exists with a native C interface as *LAPACKE_zsytrs_aa*.

4.6.192 zsytrs_rook

ZSYTRS_ROOK solves a system of linear equations A*X = B with a complex symmetric matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by ZSYTRF_ROOK.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsytrs_rook(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void zsytrs_rook_(const char *uplo, const armpl_int_t *n,
                  const armpl_int_t *nrhs, const armpl_doublecomplex_t *a,
                  const armpl_int_t *lda, const armpl_int_t *ipiv,
                  armpl_doublecomplex_t *b, const armpl_int_t *ldb,
                  armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by ZSYTRF_ROOK.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZSYTRF_ROOK.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csytrs_rook*, *dsytrs_rook* and *ssytrs_rook*. It also exists with a native C interface as *LAPACKE_zsytrs_rook*.

4.6.193 ztbtrs

ztbtrs solves a triangular system of the form

```
A * X = B,  A**T * X = B,  or  A**H * X = B,
```

where A is a triangular band matrix of order N, and B is an N-by-NRHS matrix. A check is made to verify that A is nonsingular.

Syntax

Fortran specification:

```
use armpl_library

subroutine ztbtrs(UPLO, TRANS, DIAG, N, KD, NRHS, AB, LDAB, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void ztbtrs_(const char *uplo, const char *trans, const char *diag,
             const armpl_int_t *n, const armpl_int_t *kd,
             const armpl_int_t *nrhs, const armpl_doublecomplex_t *ab,
             const armpl_int_t *ldab, armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose)

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals or subdiagonals of the triangular band matrix A. KD >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AB Input parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). The upper or lower triangular band matrix A, stored in the first kd+1 rows of AB. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd). If DIAG = 'U', the diagonal elements of A are not referenced and are assumed to be 1.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, if INFO = 0, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element of A is zero, indicating that the matrix is singular and the solutions X have not been computed.

Related Information

For this routine in other precisions, please see *ctbtrs*, *dtbtrs* and *stbtrs*. It also exists with a native C interface as *LAPACKE_ztbtrs*.

4.6.194 ztptrs

ztptrs solves a triangular system of the form

```
A * X = B,  A**T * X = B,  or  A**H * X = B,
```

where A is a triangular matrix of order N stored in packed format, and B is an N-by-NRHS matrix. A check is made to verify that A is nonsingular.

Syntax

Fortran specification:

```
use armpl_library

subroutine ztptrs(UPLO, TRANS, DIAG, N, NRHS, AP, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void ztptrs_(const char *uplo, const char *trans, const char *diag,
             const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_doublecomplex_t *ap, armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose)

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AP Input parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). The upper or lower triangular matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, if INFO = 0, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element of A is zero, indicating that the matrix is singular and the solutions X have not been computed.

Related Information

For this routine in other precisions, please see *ctptrs*, *dtptrs* and *stptrs*. It also exists with a native C interface as *LAPACKE_ztptrs*.

4.7 LAPACK matrix equilibration routines

4.7.1 cgbequ

cgbequ computes row and column scalings intended to equilibrate an M-by-N band matrix A and reduce its condition number. R returns the row scale factors and C the column scale factors, chosen to try to make the largest element in each row and column of the matrix B with elements B(i,j)=R(i)*A(i,j)*C(j) have absolute value 1.

R(i) and C(j) are restricted to be between SMLNUM = smallest safe number and BIGNUM = largest safe number. Use of these scaling factors is not guaranteed to reduce the condition number of A but works well in practice.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgbequ(M, N, KL, KU, AB, LDAB, R, C, ROWCND, COLCND, AMAX, INFO)
```

C specification:

```
#include "armpl.h"

void cgbequ_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *kl, const armpl_int_t *ku,
             const armpl_singlecomplex_t *ab, const armpl_int_t *ldab,
             float *r, float *c, float *rowcnd, float *colcnd, float *amax,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

AB Input parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). The band matrix A, stored in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(ku+1+i-j,j) = A(i,j) for max(1,j-ku)<=i<=min(m,j+kl).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KL+KU+1.

R Output parameter.

R is REAL

R is an array, dimension (M). If INFO = 0, or INFO > M, R contains the row scale factors for A.

C Output parameter.

C is REAL

C is an array, dimension (N). If INFO = 0, C contains the column scale factors for A.

ROWCND Output parameter.

ROWCND is REAL

If INFO = 0 or INFO > M, ROWCND contains the ratio of the smallest R(i) to the largest R(i). If ROWCND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by R.

COLCND Output parameter.

COLCND is REAL

If INFO = 0, COLCND contains the ratio of the smallest C(i) to the largest C(i). If COLCND >= 0.1, it is not worth scaling by C.

AMAX Output parameter.

AMAX is REAL

Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= M: the i-th row of A is exactly zero > M: the (i-M)-th column of A is exactly zero

Related Information

For this routine in other precisions, please see *dgbequ*, *sgbequ* and *zgbequ*. It also exists with a native C interface as *LAPACKE_cgbequ*.

4.7.2 cgbequb

cgbequb computes row and column scalings intended to equilibrate an M-by-N matrix A and reduce its condition number. R returns the row scale factors and C the column scale factors, chosen to try to make the largest element in each row and column of the matrix B with elements B(i,j)=R(i)*A(i,j)*C(j) have an absolute value of at most the radix.

R(i) and C(j) are restricted to be a power of the radix between SMLNUM = smallest safe number and BIGNUM = largest safe number. Use of these scaling factors is not guaranteed to reduce the condition number of A but works well in practice.

This routine differs from CGEEQU by restricting the scaling factors to a power of the radix. Barring over- and underflow, scaling by these factors introduces no additional rounding errors. However, the scaled entries' magnitudes are no longer approximately 1 but lie between sqrt(radix) and 1/sqrt(radix).

Syntax

Fortran specification:

```
use armpl_library

subroutine cgbequb(M, N, KL, KU, AB, LDAB, R, C, ROWCND, COLCND, AMAX, INFO)
```

C specification:

```
#include "armpl.h"

void cgbequb_(const armpl_int_t *m, const armpl_int_t *n,
              const armpl_int_t *kl, const armpl_int_t *ku,
              const armpl_singlecomplex_t *ab, const armpl_int_t *ldab,
```

(continues on next page)

(continued from previous page)

```
                float *r, float *c, float *rowcnd, float *colcnd, float *amax,
                armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

AB Input parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(KU+1+i-j,j) = A(i,j) for max(1,j-KU)<=i<=min(N,j+kl)

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array A. LDAB >= max(1, M).

R Output parameter.

R is REAL

R is an array, dimension (M). If INFO = 0 or INFO > M, R contains the row scale factors for A.

C Output parameter.

C is REAL

C is an array, dimension (N). If INFO = 0, C contains the column scale factors for A.

ROWCND Output parameter.

ROWCND is REAL

If INFO = 0 or INFO > M, ROWCND contains the ratio of the smallest R(i) to the largest R(i). If ROWCND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by R.

COLCND Output parameter.

COLCND is REAL

If INFO = 0, COLCND contains the ratio of the smallest C(i) to the largest C(i). If COLCND >= 0.1, it is not worth scaling by C.

AMAX Output parameter.

AMAX is REAL

Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= M: the i-th row of A is exactly zero > M: the (i-M)-th column of A is exactly zero

Related Information

For this routine in other precisions, please see *dgbequb*, *sgbequb* and *zgbequb*. It also exists with a native C interface as *LAPACKE_cgbequb*.

4.7.3 cgeequ

cgeequ computes row and column scalings intended to equilibrate an M-by-N matrix A and reduce its condition number. R returns the row scale factors and C the column scale factors, chosen to try to make the largest element in each row and column of the matrix B with elements B(i,j)=R(i)*A(i,j)*C(j) have absolute value 1.

R(i) and C(j) are restricted to be between SMLNUM = smallest safe number and BIGNUM = largest safe number. Use of these scaling factors is not guaranteed to reduce the condition number of A but works well in practice.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgeequ(M, N, A, LDA, R, C, ROWCND, COLCND, AMAX, INFO)
```

C specification:

```
#include "armpl.h"

void cgeequ_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda, float *r,
             float *c, float *rowcnd, float *colcnd, float *amax,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The M-by-N matrix whose equilibration factors are to be computed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

R Output parameter.

R is REAL

R is an array, dimension (M). If INFO = 0 or INFO > M, R contains the row scale factors for A.

C Output parameter.

C is REAL

C is an array, dimension (N). If INFO = 0, C contains the column scale factors for A.

ROWCND Output parameter.

ROWCND is REAL

If INFO = 0 or INFO > M, ROWCND contains the ratio of the smallest R(i) to the largest R(i). If ROWCND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by R.

COLCND Output parameter.

COLCND is REAL

If INFO = 0, COLCND contains the ratio of the smallest C(i) to the largest C(i). If COLCND >= 0.1, it is not worth scaling by C.

AMAX Output parameter.

AMAX is REAL

Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= M: the i-th row of A is exactly zero > M: the (i-M)-th column of A is exactly zero

Related Information

For this routine in other precisions, please see *dgeequ*, *sgeequ* and *zgeequ*. It also exists with a native C interface as *LAPACKE_cgeequ*.

4.7.4 cgeequb

cgeequb computes row and column scalings intended to equilibrate an M-by-N matrix A and reduce its condition number. R returns the row scale factors and C the column scale factors, chosen to try to make the largest element in each row and column of the matrix B with elements B(i,j)=R(i)*A(i,j)*C(j) have an absolute value of at most the radix.

R(i) and C(j) are restricted to be a power of the radix between SMLNUM = smallest safe number and BIGNUM = largest safe number. Use of these scaling factors is not guaranteed to reduce the condition number of A but works well in practice.

This routine differs from CGEEQU by restricting the scaling factors to a power of the radix. Barring over- and underflow, scaling by these factors introduces no additional rounding errors. However, the scaled entries' magnitudes are no longer approximately 1 but lie between sqrt(radix) and 1/sqrt(radix).

Syntax

Fortran specification:

```
use armpl_library

subroutine cgeequb(M, N, A, LDA, R, C, ROWCND, COLCND, AMAX, INFO)
```

C specification:

```
#include "armpl.h"

void cgeequb_(const armpl_int_t *m, const armpl_int_t *n,
              const armpl_singlecomplex_t *a, const armpl_int_t *lda,
              float *r, float *c, float *rowcnd, float *colcnd, float *amax,
              armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The M-by-N matrix whose equilibration factors are to be computed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

R Output parameter.

R is REAL

R is an array, dimension (M). If INFO = 0 or INFO > M, R contains the row scale factors for A.

C Output parameter.

C is REAL

C is an array, dimension (N). If INFO = 0, C contains the column scale factors for A.

ROWCND Output parameter.

ROWCND is REAL

If INFO = 0 or INFO > M, ROWCND contains the ratio of the smallest R(i) to the largest R(i). If ROWCND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by R.

COLCND Output parameter.

COLCND is REAL

If INFO = 0, COLCND contains the ratio of the smallest C(i) to the largest C(i). If COLCND >= 0.1, it is not worth scaling by C.

AMAX Output parameter.

AMAX is REAL

Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= M: the i-th row of A is exactly zero > M: the (i-M)-th column of A is exactly zero

Related Information

For this routine in other precisions, please see *dgeequb*, *sgeequb* and *zgeequb*. It also exists with a native C interface as *LAPACKE_cgeequb*.

4.7.5 cheequb

cheequb computes row and column scalings intended to equilibrate a Hermitian matrix A (with respect to the Euclidean norm) and reduce its condition number. The scale factors S are computed by the BIN algorithm (see references) so that the scaled matrix B with elements B(i,j) = S(i)*A(i,j)*S(j) has a condition number within a factor N of the smallest possible condition number over all possible diagonal scalings.

Syntax

Fortran specification:

```
use armpl_library

subroutine cheequb(UPLO, N, A, LDA, S, SCOND, AMAX, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void cheequb_(char *uplo, const armpl_int_t *n,
              const armpl_singlecomplex_t *a, const armpl_int_t *lda,
              float *s, float *scond, float *amax,
              armpl_singlecomplex_t *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The N-by-N Hermitian matrix whose scaling factors are to be computed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

S Output parameter.

S is REAL

S is an array, dimension (N). If INFO = 0, S contains the scale factors for A.

SCOND Output parameter.

SCOND is REAL

If INFO = 0, S contains the ratio of the smallest S(i) to the largest S(i). If SCOND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by S.

AMAX Output parameter.

AMAX is REAL

Largest absolute value of any matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element is nonpositive.

Related Information

For this routine in other precisions, please see *zheequb*. It also exists with a native C interface as *LAPACKE_cheequb*.

4.7.6 cpbequ

cpbequ computes row and column scalings intended to equilibrate a Hermitian positive definite band matrix A and reduce its condition number (with respect to the two-norm). S contains the scale factors, S(i) = 1/sqrt(A(i,i)), chosen so that the scaled matrix B with elements B(i,j) = S(i)*A(i,j)*S(j) has ones on the diagonal. This choice of S puts the condition number of B within a factor N of the smallest possible condition number over all possible diagonal scalings.

Syntax

Fortran specification:

```
use armpl_library

subroutine cpbequ(UPLO, N, KD, AB, LDAB, S, SCOND, AMAX, INFO)
```

C specification:

```
#include "armpl.h"

void cpbequ_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
             const armpl_singlecomplex_t *ab, const armpl_int_t *ldab,
             float *s, float *scond, float *amax, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangular of A is stored; = 'L': Lower triangular of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). The upper or lower triangle of the Hermitian band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array A. LDAB >= KD+1.

S Output parameter.

S is REAL

S is an array, dimension (N). If INFO = 0, S contains the scale factors for A.

SCOND Output parameter.

SCOND is REAL

If INFO = 0, S contains the ratio of the smallest S(i) to the largest S(i). If SCOND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by S.

AMAX Output parameter.

AMAX is REAL

Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, the i-th diagonal element is nonpositive.

Related Information

For this routine in other precisions, please see *dpbequ*, *spbequ* and *zpbequ*. It also exists with a native C interface as *LAPACKE_cpbequ*.

4.7.7 cpoequ

`cpoequ` computes row and column scalings intended to equilibrate a Hermitian positive definite matrix A and reduce its condition number (with respect to the two-norm). S contains the scale factors, S(i) = 1/sqrt(A(i,i)), chosen so that the scaled matrix B with elements B(i,j) = S(i)*A(i,j)*S(j) has ones on the diagonal. This choice of S puts the condition number of B within a factor N of the smallest possible condition number over all possible diagonal scalings.

Syntax

Fortran specification:

```
use armpl_library

subroutine cpoequ(N, A, LDA, S, SCOND, AMAX, INFO)
```

C specification:

```
#include "armpl.h"

void cpoequ_(const armpl_int_t *n, const armpl_singlecomplex_t *a,
             const armpl_int_t *lda, float *s, float *scond, float *amax,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The N-by-N Hermitian positive definite matrix whose scaling factors are to be computed. Only the diagonal elements of A are referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

S Output parameter.

S is REAL

S is an array, dimension (N). If INFO = 0, S contains the scale factors for A.

SCOND Output parameter.

SCOND is REAL

If INFO = 0, S contains the ratio of the smallest S(i) to the largest S(i). If SCOND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by S.

AMAX Output parameter.

AMAX is REAL

Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element is nonpositive.

Related Information

For this routine in other precisions, please see *dpoequ*, *spoequ* and *zpoequ*. It also exists with a native C interface as *LAPACKE_cpoequ*.

4.7.8 cpoequb

cpoequb computes row and column scalings intended to equilibrate a Hermitian positive definite matrix A and reduce its condition number (with respect to the two-norm). S contains the scale factors, S(i) = 1/sqrt(A(i,i)), chosen so that the scaled matrix B with elements B(i,j) = S(i)*A(i,j)*S(j) has ones on the diagonal. This choice of S puts the condition number of B within a factor N of the smallest possible condition number over all possible diagonal scalings.

This routine differs from CPOEQU by restricting the scaling factors to a power of the radix. Barring over- and underflow, scaling by these factors introduces no additional rounding errors. However, the scaled diagonal entries are no longer approximately 1 but lie between sqrt(radix) and 1/sqrt(radix).

Syntax

Fortran specification:

```
use armpl_library

subroutine cpoequb(N, A, LDA, S, SCOND, AMAX, INFO)
```

C specification:

```
#include "armpl.h"

void cpoequb_(const armpl_int_t *n, const armpl_singlecomplex_t *a,
              const armpl_int_t *lda, float *s, float *scond, float *amax,
              armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The N-by-N Hermitian positive definite matrix whose scaling factors are to be computed. Only the diagonal elements of A are referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

S Output parameter.

S is REAL

S is an array, dimension (N). If INFO = 0, S contains the scale factors for A.

SCOND Output parameter.

SCOND is REAL

If INFO = 0, S contains the ratio of the smallest S(i) to the largest S(i). If SCOND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by S.

AMAX Output parameter.

AMAX is REAL

Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element is nonpositive.

Related Information

For this routine in other precisions, please see *dpoequb*, *spoequb* and *zpoequb*. It also exists with a native C interface as *LAPACKE_cpoequb*.

4.7.9 cppequ

cppequ computes row and column scalings intended to equilibrate a Hermitian positive definite matrix A in packed storage and reduce its condition number (with respect to the two-norm). S contains the scale factors, S(i)=1/sqrt(A(i,i)), chosen so that the scaled matrix B with elements B(i,j)=S(i)*A(i,j)*S(j) has ones on the diagonal. This choice of S puts the condition number of B within a factor N of the smallest possible condition number over all possible diagonal scalings.

Syntax

Fortran specification:

```
use armpl_library

subroutine cppequ(UPLO, N, AP, S, SCOND, AMAX, INFO)
```

C specification:

```
#include "armpl.h"

void cppequ_(const char *uplo, const armpl_int_t *n,
             const armpl_singlecomplex_t *ap, float *s, float *scond,
             float *amax, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). The upper or lower triangle of the Hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

S Output parameter.

S is REAL

S is an array, dimension (N). If INFO = 0, S contains the scale factors for A.

SCOND Output parameter.

SCOND is REAL

If INFO = 0, S contains the ratio of the smallest S(i) to the largest S(i). If SCOND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by S.

AMAX Output parameter.

AMAX is REAL

Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element is nonpositive.

Related Information

For this routine in other precisions, please see *dppequ*, *sppequ* and *zppequ*. It also exists with a native C interface as *LAPACKE_cppequ*.

4.7.10 csyequb

csyequb computes row and column scalings intended to equilibrate a symmetric matrix A (with respect to the Euclidean norm) and reduce its condition number. The scale factors S are computed by the BIN algorithm (see references) so that the scaled matrix B with elements B(i,j) = S(i)*A(i,j)*S(j) has a condition number within a factor N of the smallest possible condition number over all possible diagonal scalings.

Syntax

Fortran specification:

```
use armpl_library

subroutine csyequb(UPLO, N, A, LDA, S, SCOND, AMAX, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void csyequb_(char *uplo, const armpl_int_t *n,
              const armpl_singlecomplex_t *a, const armpl_int_t *lda,
              float *s, float *scond, float *amax,
              armpl_singlecomplex_t *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The N-by-N symmetric matrix whose scaling factors are to be computed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

S Output parameter.

S is REAL

S is an array, dimension (N). If INFO = 0, S contains the scale factors for A.

SCOND Output parameter.

SCOND is REAL

If INFO = 0, S contains the ratio of the smallest S(i) to the largest S(i). If SCOND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by S.

AMAX Output parameter.

AMAX is REAL

Largest absolute value of any matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element is nonpositive.

Related Information

For this routine in other precisions, please see *dsyequb*, *ssyequb* and *zsyequb*. It also exists with a native C interface as *LAPACKE_csyequb*.

4.7.11 dgbequ

dgbequ computes row and column scalings intended to equilibrate an M-by-N band matrix A and reduce its condition number. R returns the row scale factors and C the column scale factors, chosen to try to make the largest element in each row and column of the matrix B with elements B(i,j)=R(i)*A(i,j)*C(j) have absolute value 1.

R(i) and C(j) are restricted to be between SMLNUM = smallest safe number and BIGNUM = largest safe number. Use of these scaling factors is not guaranteed to reduce the condition number of A but works well in practice.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgbequ(M, N, KL, KU, AB, LDAB, R, C, ROWCND, COLCND, AMAX, INFO)
```

C specification:

```
#include "armpl.h"

void dgbequ_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *kl, const armpl_int_t *ku, const double *ab,
             const armpl_int_t *ldab, double *r, double *c, double *rowcnd,
             double *colcnd, double *amax, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

AB Input parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). The band matrix A, stored in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(ku+1+i-j,j) = A(i,j) for max(1,j-ku)<=i<=min(m,j+kl).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KL+KU+1.

R Output parameter.

R is DOUBLE PRECISION

R is an array, dimension (M). If INFO = 0, or INFO > M, R contains the row scale factors for A.

C Output parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). If INFO = 0, C contains the column scale factors for A.

ROWCND Output parameter.

ROWCND is DOUBLE PRECISION

If INFO = 0 or INFO > M, ROWCND contains the ratio of the smallest R(i) to the largest R(i). If ROWCND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by R.

COLCND Output parameter.

COLCND is DOUBLE PRECISION

If INFO = 0, COLCND contains the ratio of the smallest C(i) to the largest C(i). If COLCND >= 0.1, it is not worth scaling by C.

AMAX Output parameter.

AMAX is DOUBLE PRECISION

Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= M: the i-th row of A is exactly zero > M: the (i-M)-th column of A is exactly zero

Related Information

For this routine in other precisions, please see *cgbequ*, *sgbequ* and *zgbequ*. It also exists with a native C interface as *LAPACKE_dgbequ*.

4.7.12 dgbequb

dgbequb computes row and column scalings intended to equilibrate an M-by-N matrix A and reduce its condition number. R returns the row scale factors and C the column scale factors, chosen to try to make the largest element in each row and column of the matrix B with elements B(i,j)=R(i)*A(i,j)*C(j) have an absolute value of at most the radix.

R(i) and C(j) are restricted to be a power of the radix between SMLNUM = smallest safe number and BIGNUM = largest safe number. Use of these scaling factors is not guaranteed to reduce the condition number of A but works well in practice.

This routine differs from DGEEQU by restricting the scaling factors to a power of the radix. Barring over- and underflow, scaling by these factors introduces no additional rounding errors. However, the scaled entries' magnitudes are no longer approximately 1 but lie between sqrt(radix) and 1/sqrt(radix).

Syntax

Fortran specification:

```
use armpl_library

subroutine dgbequb(M, N, KL, KU, AB, LDAB, R, C, ROWCND, COLCND, AMAX, INFO)
```

C specification:

```
#include "armpl.h"

void dgbequb_(const armpl_int_t *m, const armpl_int_t *n,
              const armpl_int_t *kl, const armpl_int_t *ku, const double *ab,
              const armpl_int_t *ldab, double *r, double *c, double *rowcnd,
              double *colcnd, double *amax, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

AB Input parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(KU+1+i-j,j) = A(i,j) for max(1,j-KU)<=i<=min(N,j+kl)

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array A. LDAB >= max(1, M).

R Output parameter.

R is DOUBLE PRECISION

R is an array, dimension (M). If INFO = 0 or INFO > M, R contains the row scale factors for A.

C Output parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). If INFO = 0, C contains the column scale factors for A.

ROWCND Output parameter.

ROWCND is DOUBLE PRECISION

If INFO = 0 or INFO > M, ROWCND contains the ratio of the smallest R(i) to the largest R(i). If ROWCND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by R.

COLCND Output parameter.

COLCND is DOUBLE PRECISION

If INFO = 0, COLCND contains the ratio of the smallest C(i) to the largest C(i). If COLCND >= 0.1, it is not worth scaling by C.

AMAX Output parameter.

AMAX is DOUBLE PRECISION

Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= M: the i-th row of A is exactly zero > M: the (i-M)-th column of A is exactly zero

Related Information

For this routine in other precisions, please see *cgbequb*, *sgbequb* and *zgbequb*. It also exists with a native C interface as *LAPACKE_dgbequb*.

4.7.13 dgeequ

dgeequ computes row and column scalings intended to equilibrate an M-by-N matrix A and reduce its condition number. R returns the row scale factors and C the column scale factors, chosen to try to make the largest element in each row and column of the matrix B with elements B(i,j)=R(i)*A(i,j)*C(j) have absolute value 1.

R(i) and C(j) are restricted to be between SMLNUM = smallest safe number and BIGNUM = largest safe number. Use of these scaling factors is not guaranteed to reduce the condition number of A but works well in practice.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgeequ(M, N, A, LDA, R, C, ROWCND, COLCND, AMAX, INFO)
```

C specification:

```
#include "armpl.h"

void dgeequ_(const armpl_int_t *m, const armpl_int_t *n, const double *a,
             const armpl_int_t *lda, double *r, double *c, double *rowcnd,
             double *colcnd, double *amax, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). The M-by-N matrix whose equilibration factors are to be computed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

R Output parameter.

R is DOUBLE PRECISION

R is an array, dimension (M). If INFO = 0 or INFO > M, R contains the row scale factors for A.

C Output parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). If INFO = 0, C contains the column scale factors for A.

ROWCND Output parameter.

ROWCND is DOUBLE PRECISION

If INFO = 0 or INFO > M, ROWCND contains the ratio of the smallest R(i) to the largest R(i). If ROWCND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by R.

COLCND Output parameter.

COLCND is DOUBLE PRECISION

If INFO = 0, COLCND contains the ratio of the smallest C(i) to the largest C(i). If COLCND >= 0.1, it is not worth scaling by C.

AMAX Output parameter.

AMAX is DOUBLE PRECISION

Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= M: the i-th row of A is exactly zero > M: the (i-M)-th column of A is exactly zero

Related Information

For this routine in other precisions, please see *cgeequ*, *sgeequ* and *zgeequ*. It also exists with a native C interface as *LAPACKE_dgeequ*.

4.7.14 dgeequb

`dgeequb` computes row and column scalings intended to equilibrate an M-by-N matrix A and reduce its condition number. R returns the row scale factors and C the column scale factors, chosen to try to make the largest element in each row and column of the matrix B with elements B(i,j)=R(i)*A(i,j)*C(j) have an absolute value of at most the radix.

R(i) and C(j) are restricted to be a power of the radix between SMLNUM = smallest safe number and BIGNUM = largest safe number. Use of these scaling factors is not guaranteed to reduce the condition number of A but works well in practice.

This routine differs from DGEEQU by restricting the scaling factors to a power of the radix. Barring over- and underflow, scaling by these factors introduces no additional rounding errors. However, the scaled entries' magnitudes are no longer approximately 1 but lie between sqrt(radix) and 1/sqrt(radix).

Syntax

Fortran specification:

```
use armpl_library

subroutine dgeequb(M, N, A, LDA, R, C, ROWCND, COLCND, AMAX, INFO)
```

C specification:

```
#include "armpl.h"

void dgeequb_(const armpl_int_t *m, const armpl_int_t *n, const double *a,
              const armpl_int_t *lda, double *r, double *c, double *rowcnd,
              double *colcnd, double *amax, armpl_int_t *info);
```

Parameters

M Input parameter.

`M` is `INTEGER`

The number of rows of the matrix `A`. `M` >= 0.

N Input parameter.

`N` is `INTEGER`

The number of columns of the matrix `A`. `N` >= 0.

A Input parameter.

`A` is `DOUBLE PRECISION`

`A` is an array, dimension (`LDA`, `N`). The M-by-N matrix whose equilibration factors are to be computed.

LDA Input parameter.

`LDA` is `INTEGER`

The leading dimension of the array `A`. `LDA` >= max(1, `M`).

R Output parameter.

`R` is `DOUBLE PRECISION`

`R` is an array, dimension (`M`). If `INFO` = 0 or `INFO` > `M`, `R` contains the row scale factors for `A`.

C Output parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). If INFO = 0, C contains the column scale factors for A.

ROWCND Output parameter.

ROWCND is DOUBLE PRECISION

If INFO = 0 or INFO > M, ROWCND contains the ratio of the smallest R(i) to the largest R(i). If ROWCND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by R.

COLCND Output parameter.

COLCND is DOUBLE PRECISION

If INFO = 0, COLCND contains the ratio of the smallest C(i) to the largest C(i). If COLCND >= 0.1, it is not worth scaling by C.

AMAX Output parameter.

AMAX is DOUBLE PRECISION

Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= M: the i-th row of A is exactly zero > M: the (i-M)-th column of A is exactly zero

Related Information

For this routine in other precisions, please see *cgeequb*, *sgeequb* and *zgeequb*. It also exists with a native C interface as *LAPACKE_dgeequb*.

4.7.15 dpbequ

dpbequ computes row and column scalings intended to equilibrate a symmetric positive definite band matrix A and reduce its condition number (with respect to the two-norm). S contains the scale factors, S(i) = 1/sqrt(A(i,i)), chosen so that the scaled matrix B with elements B(i,j) = S(i)*A(i,j)*S(j) has ones on the diagonal. This choice of S puts the condition number of B within a factor N of the smallest possible condition number over all possible diagonal scalings.

Syntax

Fortran specification:

```
use armpl_library

subroutine dpbequ(UPLO, N, KD, AB, LDAB, S, SCOND, AMAX, INFO)
```

C specification:

```
#include "armpl.h"

void dpbequ_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
             const double *ab, const armpl_int_t *ldab, double *s,
             double *scond, double *amax, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangular of A is stored; = 'L': Lower triangular of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). The upper or lower triangle of the symmetric band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array A. LDAB >= KD+1.

S Output parameter.

S is DOUBLE PRECISION

S is an array, dimension (N). If INFO = 0, S contains the scale factors for A.

SCOND Output parameter.

SCOND is DOUBLE PRECISION

If INFO = 0, S contains the ratio of the smallest S(i) to the largest S(i). If SCOND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by S.

AMAX Output parameter.

AMAX is DOUBLE PRECISION

Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, the i-th diagonal element is nonpositive.

Related Information

For this routine in other precisions, please see *cpbequ*, *spbequ* and *zpbequ*. It also exists with a native C interface as *LAPACKE_dpbequ*.

4.7.16 dpoequ

dpoequ computes row and column scalings intended to equilibrate a symmetric positive definite matrix A and reduce its condition number (with respect to the two-norm). S contains the scale factors, S(i) = 1/sqrt(A(i,i)), chosen so that the scaled matrix B with elements B(i,j) = S(i)*A(i,j)*S(j) has ones on the diagonal. This choice of S puts the condition number of B within a factor N of the smallest possible condition number over all possible diagonal scalings.

Syntax

Fortran specification:

```
use armpl_library

subroutine dpoequ(N, A, LDA, S, SCOND, AMAX, INFO)
```

C specification:

```
#include "armpl.h"

void dpoequ_(const armpl_int_t *n, const double *a, const armpl_int_t *lda,
             double *s, double *scond, double *amax, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). The N-by-N symmetric positive definite matrix whose scaling factors are to be computed. Only the diagonal elements of A are referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

S Output parameter.

S is DOUBLE PRECISION

S is an array, dimension (N). If INFO = 0, S contains the scale factors for A.

SCOND Output parameter.

SCOND is DOUBLE PRECISION

If INFO = 0, S contains the ratio of the smallest S(i) to the largest S(i). If SCOND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by S.

AMAX Output parameter.

AMAX is DOUBLE PRECISION

Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element is nonpositive.

Related Information

For this routine in other precisions, please see *cpoequ*, *spoequ* and *zpoequ*. It also exists with a native C interface as *LAPACKE_dpoequ*.

4.7.17 dpoequb

dpoequb computes row and column scalings intended to equilibrate a symmetric positive definite matrix A and reduce its condition number (with respect to the two-norm). S contains the scale factors, S(i) = 1/sqrt(A(i,i)), chosen so that the scaled matrix B with elements B(i,j) = S(i)*A(i,j)*S(j) has ones on the diagonal. This choice of S puts the condition number of B within a factor N of the smallest possible condition number over all possible diagonal scalings.

This routine differs from DPOEQU by restricting the scaling factors to a power of the radix. Barring over- and underflow, scaling by these factors introduces no additional rounding errors. However, the scaled diagonal entries are no longer approximately 1 but lie between sqrt(radix) and 1/sqrt(radix).

Syntax

Fortran specification:

```
use armpl_library

subroutine dpoequb(N, A, LDA, S, SCOND, AMAX, INFO)
```

C specification:

```
#include "armpl.h"

void dpoequb_(const armpl_int_t *n, const double *a, const armpl_int_t *lda,
              double *s, double *scond, double *amax, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). The N-by-N symmetric positive definite matrix whose scaling factors are to be computed. Only the diagonal elements of A are referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

S Output parameter.

S is DOUBLE PRECISION

S is an array, dimension (N). If INFO = 0, S contains the scale factors for A.

SCOND Output parameter.

SCOND is DOUBLE PRECISION

If INFO = 0, S contains the ratio of the smallest S(i) to the largest S(i). If SCOND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by S.

AMAX Output parameter.

AMAX is DOUBLE PRECISION

Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element is nonpositive.

Related Information

For this routine in other precisions, please see *cpoequb*, *spoequb* and *zpoequb*. It also exists with a native C interface as *LAPACKE_dpoequb*.

4.7.18 dppequ

dppequ computes row and column scalings intended to equilibrate a symmetric positive definite matrix A in packed storage and reduce its condition number (with respect to the two-norm). S contains the scale factors, S(i)=1/sqrt(A(i,i)), chosen so that the scaled matrix B with elements B(i,j)=S(i)*A(i,j)*S(j) has ones on the diagonal. This choice of S puts the condition number of B within a factor N of the smallest possible condition number over all possible diagonal scalings.

Syntax

Fortran specification:

```
use armpl_library

subroutine dppequ(UPLO, N, AP, S, SCOND, AMAX, INFO)
```

C specification:

```
#include "armpl.h"

void dppequ_(const char *uplo, const armpl_int_t *n, const double *ap,
             double *s, double *scond, double *amax, armpl_int_t *info,
             ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input parameter.

AP is DOUBLE PRECISION

AP is an array, dimension (N*(N+1)/2). The upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

S Output parameter.

S is DOUBLE PRECISION

S is an array, dimension (N). If INFO = 0, S contains the scale factors for A.

SCOND Output parameter.

SCOND is DOUBLE PRECISION

If INFO = 0, S contains the ratio of the smallest S(i) to the largest S(i). If SCOND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by S.

AMAX Output parameter.

AMAX is DOUBLE PRECISION

Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element is nonpositive.

Related Information

For this routine in other precisions, please see *cppequ*, *sppequ* and *zppequ*. It also exists with a native C interface as *LAPACKE_dppequ*.

4.7.19 dsyequb

dsyequb computes row and column scalings intended to equilibrate a symmetric matrix A (with respect to the Euclidean norm) and reduce its condition number. The scale factors S are computed by the BIN algorithm (see references) so that the scaled matrix B with elements B(i,j) = S(i)*A(i,j)*S(j) has a condition number within a factor N of the smallest possible condition number over all possible diagonal scalings.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsyequb(UPLO, N, A, LDA, S, SCOND, AMAX, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dsyequb_(char *uplo, const armpl_int_t *n, const double *a,
              const armpl_int_t *lda, double *s, double *scond, double *amax,
              double *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). The N-by-N symmetric matrix whose scaling factors are to be computed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

S Output parameter.

S is DOUBLE PRECISION

S is an array, dimension (N). If INFO = 0, S contains the scale factors for A.

SCOND Output parameter.

SCOND is DOUBLE PRECISION

If INFO = 0, S contains the ratio of the smallest S(i) to the largest S(i). If SCOND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by S.

AMAX Output parameter.

AMAX is DOUBLE PRECISION

Largest absolute value of any matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element is nonpositive.

Related Information

For this routine in other precisions, please see *csyequb*, *ssyequb* and *zsyequb*. It also exists with a native C interface as *LAPACKE_dsyequb*.

4.7.20 sgbequ

sgbequ computes row and column scalings intended to equilibrate an M-by-N band matrix A and reduce its condition number. R returns the row scale factors and C the column scale factors, chosen to try to make the largest element in each row and column of the matrix B with elements B(i,j)=R(i)*A(i,j)*C(j) have absolute value 1.

R(i) and C(j) are restricted to be between SMLNUM = smallest safe number and BIGNUM = largest safe number. Use of these scaling factors is not guaranteed to reduce the condition number of A but works well in practice.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgbequ(M, N, KL, KU, AB, LDAB, R, C, ROWCND, COLCND, AMAX, INFO)
```

C specification:

```
#include "armpl.h"

void sgbequ_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *kl, const armpl_int_t *ku, const float *ab,
             const armpl_int_t *ldab, float *r, float *c, float *rowcnd,
             float *colcnd, float *amax, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

AB Input parameter.

AB is REAL

AB is an array, dimension (LDAB, N). The band matrix A, stored in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(ku+1+i-j,j) = A(i,j) for max(1,j-ku)<=i<=min(m,j+kl).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KL+KU+1.

R Output parameter.

R is REAL

R is an array, dimension (M). If INFO = 0, or INFO > M, R contains the row scale factors for A.

C Output parameter.

C is REAL

C is an array, dimension (N). If INFO = 0, C contains the column scale factors for A.

ROWCND Output parameter.

ROWCND is REAL

If INFO = 0 or INFO > M, ROWCND contains the ratio of the smallest R(i) to the largest R(i). If ROWCND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by R.

COLCND Output parameter.

COLCND is REAL

If INFO = 0, COLCND contains the ratio of the smallest C(i) to the largest C(i). If COLCND >= 0.1, it is not worth scaling by C.

AMAX Output parameter.

AMAX is REAL

Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= M: the i-th row of A is exactly zero > M: the (i-M)-th column of A is exactly zero

Related Information

For this routine in other precisions, please see *cgbequ*, *dgbequ* and *zgbequ*. It also exists with a native C interface as *LAPACKE_sgbequ*.

4.7.21 sgbequb

sgbequb computes row and column scalings intended to equilibrate an M-by-N matrix A and reduce its condition number. R returns the row scale factors and C the column scale factors, chosen to try to make the largest element in each row and column of the matrix B with elements B(i,j)=R(i)*A(i,j)*C(j) have an absolute value of at most the radix.

R(i) and C(j) are restricted to be a power of the radix between SMLNUM = smallest safe number and BIGNUM = largest safe number. Use of these scaling factors is not guaranteed to reduce the condition number of A but works well in practice.

This routine differs from SGEEQU by restricting the scaling factors to a power of the radix. Barring over- and underflow, scaling by these factors introduces no additional rounding errors. However, the scaled entries' magnitudes are no longer approximately 1 but lie between sqrt(radix) and 1/sqrt(radix).

Syntax

Fortran specification:

```
use armpl_library

subroutine sgbequb(M, N, KL, KU, AB, LDAB, R, C, ROWCND, COLCND, AMAX, INFO)
```

C specification:

```
#include "armpl.h"

void sgbequb_(const armpl_int_t *m, const armpl_int_t *n,
              const armpl_int_t *kl, const armpl_int_t *ku, const float *ab,
              const armpl_int_t *ldab, float *r, float *c, float *rowcnd,
              float *colcnd, float *amax, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

AB Input parameter.

AB is REAL

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(KU+1+i-j,j) = A(i,j) for max(1,j-KU)<=i<=min(N,j+kl)

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array A. LDAB >= max(1, M).

R Output parameter.

R is REAL

R is an array, dimension (M). If INFO = 0 or INFO > M, R contains the row scale factors for A.

C Output parameter.

C is REAL

C is an array, dimension (N). If INFO = 0, C contains the column scale factors for A.

ROWCND Output parameter.

ROWCND is REAL

If INFO = 0 or INFO > M, ROWCND contains the ratio of the smallest R(i) to the largest R(i). If ROWCND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by R.

COLCND Output parameter.

COLCND is REAL

If INFO = 0, COLCND contains the ratio of the smallest C(i) to the largest C(i). If COLCND >= 0.1, it is not worth scaling by C.

AMAX Output parameter.

AMAX is REAL

Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= M: the i-th row of A is exactly zero > M: the (i-M)-th column of A is exactly zero

Related Information

For this routine in other precisions, please see *cgbequb*, *dgbequb* and *zgbequb*. It also exists with a native C interface as *LAPACKE_sgbequb*.

4.7.22 sgeequ

sgeequ computes row and column scalings intended to equilibrate an M-by-N matrix A and reduce its condition number. R returns the row scale factors and C the column scale factors, chosen to try to make the largest element in each row and column of the matrix B with elements B(i,j)=R(i)*A(i,j)*C(j) have absolute value 1.

R(i) and C(j) are restricted to be between SMLNUM = smallest safe number and BIGNUM = largest safe number. Use of these scaling factors is not guaranteed to reduce the condition number of A but works well in practice.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgeequ(M, N, A, LDA, R, C, ROWCND, COLCND, AMAX, INFO)
```

C specification:

```
#include "armpl.h"

void sgeequ_(const armpl_int_t *m, const armpl_int_t *n, const float *a,
             const armpl_int_t *lda, float *r, float *c, float *rowcnd,
             float *colcnd, float *amax, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). The M-by-N matrix whose equilibration factors are to be computed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

R Output parameter.

R is REAL

R is an array, dimension (M). If INFO = 0 or INFO > M, R contains the row scale factors for A.

C Output parameter.

C is REAL

C is an array, dimension (N). If INFO = 0, C contains the column scale factors for A.

ROWCND Output parameter.

ROWCND is REAL

If INFO = 0 or INFO > M, ROWCND contains the ratio of the smallest R(i) to the largest R(i). If ROWCND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by R.

COLCND Output parameter.

COLCND is REAL

If INFO = 0, COLCND contains the ratio of the smallest C(i) to the largest C(i). If COLCND >= 0.1, it is not worth scaling by C.

AMAX Output parameter.

AMAX is REAL

Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= M: the i-th row of A is exactly zero > M: the (i-M)-th column of A is exactly zero

Related Information

For this routine in other precisions, please see *cgeequ*, *dgeequ* and *zgeequ*. It also exists with a native C interface as *LAPACKE_sgeequ*.

4.7.23 sgeequb

sgeequb computes row and column scalings intended to equilibrate an M-by-N matrix A and reduce its condition number. R returns the row scale factors and C the column scale factors, chosen to try to make the largest element in each row and column of the matrix B with elements B(i,j)=R(i)*A(i,j)*C(j) have an absolute value of at most the radix.

R(i) and C(j) are restricted to be a power of the radix between SMLNUM = smallest safe number and BIGNUM = largest safe number. Use of these scaling factors is not guaranteed to reduce the condition number of A but works well in practice.

This routine differs from SGEEQU by restricting the scaling factors to a power of the radix. Barring over- and underflow, scaling by these factors introduces no additional rounding errors. However, the scaled entries' magnitudes are no longer approximately 1 but lie between sqrt(radix) and 1/sqrt(radix).

Syntax

Fortran specification:

```
use armpl_library

subroutine sgeequb(M, N, A, LDA, R, C, ROWCND, COLCND, AMAX, INFO)
```

C specification:

```
#include "armpl.h"

void sgeequb_(const armpl_int_t *m, const armpl_int_t *n, const float *a,
              const armpl_int_t *lda, float *r, float *c, float *rowcnd,
              float *colcnd, float *amax, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). The M-by-N matrix whose equilibration factors are to be computed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

R Output parameter.

R is REAL

R is an array, dimension (M). If INFO = 0 or INFO > M, R contains the row scale factors for A.

C Output parameter.

C is REAL

C is an array, dimension (N). If INFO = 0, C contains the column scale factors for A.

ROWCND Output parameter.

ROWCND is REAL

If INFO = 0 or INFO > M, ROWCND contains the ratio of the smallest R(i) to the largest R(i). If ROWCND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by R.

COLCND Output parameter.

COLCND is REAL

If INFO = 0, COLCND contains the ratio of the smallest C(i) to the largest C(i). If COLCND >= 0.1, it is not worth scaling by C.

AMAX Output parameter.

AMAX is REAL

Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= M: the i-th row of A is exactly zero > M: the (i-M)-th column of A is exactly zero

Related Information

For this routine in other precisions, please see *cgeequb*, *dgeequb* and *zgeequb*. It also exists with a native C interface as *LAPACKE_sgeequb*.

4.7.24 spbequ

spbequ computes row and column scalings intended to equilibrate a symmetric positive definite band matrix A and reduce its condition number (with respect to the two-norm). S contains the scale factors, S(i) = 1/sqrt(A(i,i)), chosen so that the scaled matrix B with elements B(i,j) = S(i)*A(i,j)*S(j) has ones on the diagonal. This choice of S puts the condition number of B within a factor N of the smallest possible condition number over all possible diagonal scalings.

Syntax

Fortran specification:

```
use armpl_library

subroutine spbequ(UPLO, N, KD, AB, LDAB, S, SCOND, AMAX, INFO)
```

C specification:

```
#include "armpl.h"

void spbequ_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
             const float *ab, const armpl_int_t *ldab, float *s, float *scond,
             float *amax, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangular of A is stored; = 'L': Lower triangular of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input parameter.

AB is REAL

AB is an array, dimension (LDAB, N). The upper or lower triangle of the symmetric band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array A. LDAB >= KD+1.

S Output parameter.

S is REAL

S is an array, dimension (N). If INFO = 0, S contains the scale factors for A.

SCOND Output parameter.

SCOND is REAL

If INFO = 0, S contains the ratio of the smallest S(i) to the largest S(i). If SCOND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by S.

AMAX Output parameter.

AMAX is REAL

Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, the i-th diagonal element is nonpositive.

Related Information

For this routine in other precisions, please see *cpbequ*, *dpbequ* and *zpbequ*. It also exists with a native C interface as *LAPACKE_spbequ*.

4.7.25 spoequ

spoequ computes row and column scalings intended to equilibrate a symmetric positive definite matrix A and reduce its condition number (with respect to the two-norm). S contains the scale factors, S(i) = 1/sqrt(A(i,i)), chosen so that the scaled matrix B with elements B(i,j) = S(i)*A(i,j)*S(j) has ones on the diagonal. This choice of S puts the condition number of B within a factor N of the smallest possible condition number over all possible diagonal scalings.

Syntax

Fortran specification:

```
use armpl_library

subroutine spoequ(N, A, LDA, S, SCOND, AMAX, INFO)
```

C specification:

```
#include "armpl.h"

void spoequ_(const armpl_int_t *n, const float *a, const armpl_int_t *lda,
             float *s, float *scond, float *amax, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). The N-by-N symmetric positive definite matrix whose scaling factors are to be computed. Only the diagonal elements of A are referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

S Output parameter.

S is REAL

S is an array, dimension (N). If INFO = 0, S contains the scale factors for A.

SCOND Output parameter.

SCOND is REAL

If INFO = 0, S contains the ratio of the smallest S(i) to the largest S(i). If SCOND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by S.

AMAX Output parameter.

AMAX is REAL

Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element is nonpositive.

Related Information

For this routine in other precisions, please see *cpoequ*, *dpoequ* and *zpoequ*. It also exists with a native C interface as *LAPACKE_spoequ*.

4.7.26 spoequb

spoequb computes row and column scalings intended to equilibrate a symmetric positive definite matrix A and reduce its condition number (with respect to the two-norm). S contains the scale factors, S(i) = 1/sqrt(A(i,i)), chosen so that the scaled matrix B with elements B(i,j) = S(i)*A(i,j)*S(j) has ones on the diagonal. This choice of S puts the condition number of B within a factor N of the smallest possible condition number over all possible diagonal scalings.

This routine differs from SPOEQU by restricting the scaling factors to a power of the radix. Barring over- and underflow, scaling by these factors introduces no additional rounding errors. However, the scaled diagonal entries are no longer approximately 1 but lie between sqrt(radix) and 1/sqrt(radix).

Syntax

Fortran specification:

```
use armpl_library

subroutine spoequb(N, A, LDA, S, SCOND, AMAX, INFO)
```

C specification:

```
#include "armpl.h"

void spoequb_(const armpl_int_t *n, const float *a, const armpl_int_t *lda,
              float *s, float *scond, float *amax, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). The N-by-N symmetric positive definite matrix whose scaling factors are to be computed. Only the diagonal elements of A are referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

S Output parameter.

S is REAL

S is an array, dimension (N). If INFO = 0, S contains the scale factors for A.

SCOND Output parameter.

SCOND is REAL

If INFO = 0, S contains the ratio of the smallest S(i) to the largest S(i). If SCOND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by S.

AMAX Output parameter.

AMAX is REAL

Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element is nonpositive.

Related Information

For this routine in other precisions, please see *cpoequb*, *dpoequb* and *zpoequb*. It also exists with a native C interface as *LAPACKE_spoequb*.

4.7.27 sppequ

sppequ computes row and column scalings intended to equilibrate a symmetric positive definite matrix A in packed storage and reduce its condition number (with respect to the two-norm). S contains the scale factors, S(i)=1/sqrt(A(i,i)), chosen so that the scaled matrix B with elements B(i,j)=S(i)*A(i,j)*S(j) has ones on the diagonal. This choice of S puts the condition number of B within a factor N of the smallest possible condition number over all possible diagonal scalings.

Syntax

Fortran specification:

```
use armpl_library

subroutine sppequ(UPLO, N, AP, S, SCOND, AMAX, INFO)
```

C specification:

```
#include "armpl.h"

void sppequ_(const char *uplo, const armpl_int_t *n, const float *ap,
             float *s, float *scond, float *amax, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input parameter.

AP is REAL

AP is an array, dimension (N*(N+1)/2). The upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

S Output parameter.

S is REAL

S is an array, dimension (N). If INFO = 0, S contains the scale factors for A.

SCOND Output parameter.

SCOND is REAL

If INFO = 0, S contains the ratio of the smallest S(i) to the largest S(i). If SCOND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by S.

AMAX Output parameter.

AMAX is REAL

Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element is nonpositive.

Related Information

For this routine in other precisions, please see *cppequ*, *dppequ* and *zppequ*. It also exists with a native C interface as *LAPACKE_sppequ*.

4.7.28 ssyequb

ssyequb computes row and column scalings intended to equilibrate a symmetric matrix A (with respect to the Euclidean norm) and reduce its condition number. The scale factors S are computed by the BIN algorithm (see references) so that the scaled matrix B with elements B(i,j) = S(i)*A(i,j)*S(j) has a condition number within a factor N of the smallest possible condition number over all possible diagonal scalings.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssyequb(UPLO, N, A, LDA, S, SCOND, AMAX, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void ssyequb_(char *uplo, const armpl_int_t *n, const float *a,
              const armpl_int_t *lda, float *s, float *scond, float *amax,
              float *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). The N-by-N symmetric matrix whose scaling factors are to be computed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

S Output parameter.

S is REAL

S is an array, dimension (N). If INFO = 0, S contains the scale factors for A.

SCOND Output parameter.

SCOND is REAL

If INFO = 0, S contains the ratio of the smallest S(i) to the largest S(i). If SCOND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by S.

AMAX Output parameter.

AMAX is REAL

Largest absolute value of any matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element is nonpositive.

Related Information

For this routine in other precisions, please see *csyequb*, *dsyequb* and *zsyequb*. It also exists with a native C interface as *LAPACKE_ssyequb*.

4.7.29 zgbequ

zgbequ computes row and column scalings intended to equilibrate an M-by-N band matrix A and reduce its condition number. R returns the row scale factors and C the column scale factors, chosen to try to make the largest element in each row and column of the matrix B with elements B(i,j)=R(i)*A(i,j)*C(j) have absolute value 1.

R(i) and C(j) are restricted to be between SMLNUM = smallest safe number and BIGNUM = largest safe number. Use of these scaling factors is not guaranteed to reduce the condition number of A but works well in practice.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgbequ(M, N, KL, KU, AB, LDAB, R, C, ROWCND, COLCND, AMAX, INFO)
```

C specification:

```
#include "armpl.h"

void zgbequ_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *kl, const armpl_int_t *ku,
             const armpl_doublecomplex_t *ab, const armpl_int_t *ldab,
             double *r, double *c, double *rowcnd, double *colcnd,
             double *amax, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

AB Input parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). The band matrix A, stored in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(ku+1+i-j,j) = A(i,j) for max(1,j-ku)<=i<=min(m,j+kl).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KL+KU+1.

R Output parameter.

R is DOUBLE PRECISION

R is an array, dimension (M). If INFO = 0, or INFO > M, R contains the row scale factors for A.

C Output parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). If INFO = 0, C contains the column scale factors for A.

ROWCND Output parameter.

ROWCND is DOUBLE PRECISION

If INFO = 0 or INFO > M, ROWCND contains the ratio of the smallest R(i) to the largest R(i). If ROWCND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by R.

COLCND Output parameter.

COLCND is DOUBLE PRECISION

If INFO = 0, COLCND contains the ratio of the smallest C(i) to the largest C(i). If COLCND >= 0.1, it is not worth scaling by C.

AMAX Output parameter.

AMAX is DOUBLE PRECISION

Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= M: the i-th row of A is exactly zero > M: the (i-M)-th column of A is exactly zero

Related Information

For this routine in other precisions, please see *cgbequ*, *dgbequ* and *sgbequ*. It also exists with a native C interface as *LAPACKE_zgbequ*.

4.7.30 zgbequb

zgbequb computes row and column scalings intended to equilibrate an M-by-N matrix A and reduce its condition number. R returns the row scale factors and C the column scale factors, chosen to try to make the largest element in each row and column of the matrix B with elements B(i,j)=R(i)*A(i,j)*C(j) have an absolute value of at most the radix.

R(i) and C(j) are restricted to be a power of the radix between SMLNUM = smallest safe number and BIGNUM = largest safe number. Use of these scaling factors is not guaranteed to reduce the condition number of A but works well in practice.

This routine differs from ZGEEQU by restricting the scaling factors to a power of the radix. Barring over- and underflow, scaling by these factors introduces no additional rounding errors. However, the scaled entries' magnitudes are no longer approximately 1 but lie between sqrt(radix) and 1/sqrt(radix).

Syntax

Fortran specification:

```
use armpl_library

subroutine zgbequb(M, N, KL, KU, AB, LDAB, R, C, ROWCND, COLCND, AMAX, INFO)
```

C specification:

```
#include "armpl.h"

void zgbequb_(const armpl_int_t *m, const armpl_int_t *n,
              const armpl_int_t *kl, const armpl_int_t *ku,
              const armpl_doublecomplex_t *ab, const armpl_int_t *ldab,
```

(continues on next page)

(continued from previous page)

```
                double *r, double *c, double *rowcnd, double *colcnd,
                double *amax, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

AB Input parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(KU+1+i-j,j) = A(i,j) for max(1,j-KU)<=i<=min(N,j+kl)

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array A. LDAB >= max(1, M).

R Output parameter.

R is DOUBLE PRECISION

R is an array, dimension (M). If INFO = 0 or INFO > M, R contains the row scale factors for A.

C Output parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). If INFO = 0, C contains the column scale factors for A.

ROWCND Output parameter.

ROWCND is DOUBLE PRECISION

If INFO = 0 or INFO > M, ROWCND contains the ratio of the smallest R(i) to the largest R(i). If ROWCND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by R.

COLCND Output parameter.

COLCND is DOUBLE PRECISION

If INFO = 0, COLCND contains the ratio of the smallest C(i) to the largest C(i). If COLCND >= 0.1, it is not worth scaling by C.

AMAX Output parameter.

AMAX is DOUBLE PRECISION

Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= M: the i-th row of A is exactly zero > M: the (i-M)-th column of A is exactly zero

Related Information

For this routine in other precisions, please see *cgbequb*, *dgbequb* and *sgbequb*. It also exists with a native C interface as *LAPACKE_zgbequb*.

4.7.31 zgeequ

zgeequ computes row and column scalings intended to equilibrate an M-by-N matrix A and reduce its condition number. R returns the row scale factors and C the column scale factors, chosen to try to make the largest element in each row and column of the matrix B with elements B(i,j)=R(i)*A(i,j)*C(j) have absolute value 1.

R(i) and C(j) are restricted to be between SMLNUM = smallest safe number and BIGNUM = largest safe number. Use of these scaling factors is not guaranteed to reduce the condition number of A but works well in practice.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgeequ(M, N, A, LDA, R, C, ROWCND, COLCND, AMAX, INFO)
```

C specification:

```
#include "armpl.h"

void zgeequ_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             double *r, double *c, double *rowcnd, double *colcnd,
             double *amax, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The M-by-N matrix whose equilibration factors are to be computed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

R Output parameter.

R is DOUBLE PRECISION

R is an array, dimension (M). If INFO = 0 or INFO > M, R contains the row scale factors for A.

C Output parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). If INFO = 0, C contains the column scale factors for A.

ROWCND Output parameter.

ROWCND is DOUBLE PRECISION

If INFO = 0 or INFO > M, ROWCND contains the ratio of the smallest R(i) to the largest R(i). If ROWCND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by R.

COLCND Output parameter.

COLCND is DOUBLE PRECISION

If INFO = 0, COLCND contains the ratio of the smallest C(i) to the largest C(i). If COLCND >= 0.1, it is not worth scaling by C.

AMAX Output parameter.

AMAX is DOUBLE PRECISION

Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= M: the i-th row of A is exactly zero > M: the (i-M)-th column of A is exactly zero

Related Information

For this routine in other precisions, please see *cgeequ*, *dgeequ* and *sgeequ*. It also exists with a native C interface as *LAPACKE_zgeequ*.

4.7.32 zgeequb

zgeequb computes row and column scalings intended to equilibrate an M-by-N matrix A and reduce its condition number. R returns the row scale factors and C the column scale factors, chosen to try to make the largest element in each row and column of the matrix B with elements B(i,j)=R(i)*A(i,j)*C(j) have an absolute value of at most the radix.

R(i) and C(j) are restricted to be a power of the radix between SMLNUM = smallest safe number and BIGNUM = largest safe number. Use of these scaling factors is not guaranteed to reduce the condition number of A but works well in practice.

This routine differs from ZGEEQU by restricting the scaling factors to a power of the radix. Barring over- and underflow, scaling by these factors introduces no additional rounding errors. However, the scaled entries' magnitudes are no longer approximately 1 but lie between sqrt(radix) and 1/sqrt(radix).

Syntax

Fortran specification:

```
use armpl_library

subroutine zgeequb(M, N, A, LDA, R, C, ROWCND, COLCND, AMAX, INFO)
```

C specification:

```
#include "armpl.h"

void zgeequb_(const armpl_int_t *m, const armpl_int_t *n,
              const armpl_doublecomplex_t *a, const armpl_int_t *lda,
              double *r, double *c, double *rowcnd, double *colcnd,
              double *amax, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The M-by-N matrix whose equilibration factors are to be computed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

R Output parameter.

R is DOUBLE PRECISION

R is an array, dimension (M). If INFO = 0 or INFO > M, R contains the row scale factors for A.

C Output parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). If INFO = 0, C contains the column scale factors for A.

ROWCND Output parameter.

ROWCND is DOUBLE PRECISION

If INFO = 0 or INFO > M, ROWCND contains the ratio of the smallest R(i) to the largest R(i). If ROWCND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by R.

COLCND Output parameter.

COLCND is DOUBLE PRECISION

If INFO = 0, COLCND contains the ratio of the smallest C(i) to the largest C(i). If COLCND >= 0.1, it is not worth scaling by C.

AMAX Output parameter.

AMAX is DOUBLE PRECISION

Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is <= M: the i-th row of A is exactly zero > M: the (i-M)-th column of A is exactly zero

Related Information

For this routine in other precisions, please see *cgeequb*, *dgeequb* and *sgeequb*. It also exists with a native C interface as *LAPACKE_zgeequb*.

4.7.33 zheequb

zheequb computes row and column scalings intended to equilibrate a Hermitian matrix A (with respect to the Euclidean norm) and reduce its condition number. The scale factors S are computed by the BIN algorithm (see references) so that the scaled matrix B with elements B(i,j) = S(i)*A(i,j)*S(j) has a condition number within a factor N of the smallest possible condition number over all possible diagonal scalings.

Syntax

Fortran specification:

```
use armpl_library

subroutine zheequb(UPLO, N, A, LDA, S, SCOND, AMAX, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zheequb_(char *uplo, const armpl_int_t *n,
              const armpl_doublecomplex_t *a, const armpl_int_t *lda,
              double *s, double *scond, double *amax,
              armpl_doublecomplex_t *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The N-by-N Hermitian matrix whose scaling factors are to be computed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

S Output parameter.

S is DOUBLE PRECISION

S is an array, dimension (N). If INFO = 0, S contains the scale factors for A.

SCOND Output parameter.

SCOND is DOUBLE PRECISION

If INFO = 0, S contains the ratio of the smallest S(i) to the largest S(i). If SCOND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by S.

AMAX Output parameter.

AMAX is DOUBLE PRECISION

Largest absolute value of any matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element is nonpositive.

Related Information

For this routine in other precisions, please see *cheequb*. It also exists with a native C interface as *LAPACKE_zheequb*.

4.7.34 zpbequ

zpbequ computes row and column scalings intended to equilibrate a Hermitian positive definite band matrix A and reduce its condition number (with respect to the two-norm). S contains the scale factors, S(i) = 1/sqrt(A(i,i)), chosen so that the scaled matrix B with elements B(i,j) = S(i)*A(i,j)*S(j) has ones on the diagonal. This choice of S puts the condition number of B within a factor N of the smallest possible condition number over all possible diagonal scalings.

Syntax

Fortran specification:

```
use armpl_library

subroutine zpbequ(UPLO, N, KD, AB, LDAB, S, SCOND, AMAX, INFO)
```

C specification:

```
#include "armpl.h"

void zpbequ_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
             const armpl_doublecomplex_t *ab, const armpl_int_t *ldab,
             double *s, double *scond, double *amax, armpl_int_t *info,
             ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangular of A is stored; = 'L': Lower triangular of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). The upper or lower triangle of the Hermitian band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array A. LDAB >= KD+1.

S Output parameter.

S is DOUBLE PRECISION

S is an array, dimension (N). If INFO = 0, S contains the scale factors for A.

SCOND Output parameter.

SCOND is DOUBLE PRECISION

If INFO = 0, S contains the ratio of the smallest S(i) to the largest S(i). If SCOND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by S.

AMAX Output parameter.

AMAX is DOUBLE PRECISION

Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, the i-th diagonal element is nonpositive.

Related Information

For this routine in other precisions, please see *cpbequ*, *dpbequ* and *spbequ*. It also exists with a native C interface as *LAPACKE_zpbequ*.

4.7.35 zpoequ

zpoequ computes row and column scalings intended to equilibrate a Hermitian positive definite matrix A and reduce its condition number (with respect to the two-norm). S contains the scale factors, S(i) = 1/sqrt(A(i,i)), chosen so that the scaled matrix B with elements B(i,j) = S(i)*A(i,j)*S(j) has ones on the diagonal. This choice of S puts the condition number of B within a factor N of the smallest possible condition number over all possible diagonal scalings.

Syntax

Fortran specification:

```
use armpl_library

subroutine zpoequ(N, A, LDA, S, SCOND, AMAX, INFO)
```

C specification:

```
#include "armpl.h"

void zpoequ_(const armpl_int_t *n, const armpl_doublecomplex_t *a,
             const armpl_int_t *lda, double *s, double *scond, double *amax,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The N-by-N Hermitian positive definite matrix whose scaling factors are to be computed. Only the diagonal elements of A are referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

S Output parameter.

S is DOUBLE PRECISION

S is an array, dimension (N). If INFO = 0, S contains the scale factors for A.

SCOND Output parameter.

SCOND is DOUBLE PRECISION

If INFO = 0, S contains the ratio of the smallest S(i) to the largest S(i). If SCOND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by S.

AMAX Output parameter.

AMAX is DOUBLE PRECISION

Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element is nonpositive.

Related Information

For this routine in other precisions, please see *cpoequ*, *dpoequ* and *spoequ*. It also exists with a native C interface as *LAPACKE_zpoequ*.

4.7.36 zpoequb

zpoequb computes row and column scalings intended to equilibrate a Hermitian positive definite matrix A and reduce its condition number (with respect to the two-norm). S contains the scale factors, S(i) = 1/sqrt(A(i,i)), chosen so that the scaled matrix B with elements B(i,j) = S(i)*A(i,j)*S(j) has ones on the diagonal. This choice of S puts the condition number of B within a factor N of the smallest possible condition number over all possible diagonal scalings.

This routine differs from ZPOEQU by restricting the scaling factors to a power of the radix. Barring over- and underflow, scaling by these factors introduces no additional rounding errors. However, the scaled diagonal entries are no longer approximately 1 but lie between sqrt(radix) and 1/sqrt(radix).

Syntax

Fortran specification:

```
use armpl_library

subroutine zpoequb(N, A, LDA, S, SCOND, AMAX, INFO)
```

C specification:

```
#include "armpl.h"

void zpoequb_(const armpl_int_t *n, const armpl_doublecomplex_t *a,
              const armpl_int_t *lda, double *s, double *scond, double *amax,
              armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The N-by-N Hermitian positive definite matrix whose scaling factors are to be computed. Only the diagonal elements of A are referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

S Output parameter.

S is DOUBLE PRECISION

S is an array, dimension (N). If INFO = 0, S contains the scale factors for A.

SCOND Output parameter.

SCOND is DOUBLE PRECISION

If INFO = 0, S contains the ratio of the smallest S(i) to the largest S(i). If SCOND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by S.

AMAX Output parameter.

AMAX is DOUBLE PRECISION

Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element is nonpositive.

Related Information

For this routine in other precisions, please see *cpoequb*, *dpoequb* and *spoequb*. It also exists with a native C interface as *LAPACKE_zpoequb*.

4.7.37 zppequ

zppequ computes row and column scalings intended to equilibrate a Hermitian positive definite matrix A in packed storage and reduce its condition number (with respect to the two-norm). S contains the scale factors, S(i)=1/sqrt(A(i,i)), chosen so that the scaled matrix B with elements B(i,j)=S(i)*A(i,j)*S(j) has ones on the diagonal. This choice of S puts the condition number of B within a factor N of the smallest possible condition number over all possible diagonal scalings.

Syntax

Fortran specification:

```
use armpl_library

subroutine zppequ(UPLO, N, AP, S, SCOND, AMAX, INFO)
```

C specification:

```
#include "armpl.h"

void zppequ_(const char *uplo, const armpl_int_t *n,
             const armpl_doublecomplex_t *ap, double *s, double *scond,
             double *amax, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). The upper or lower triangle of the Hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

S Output parameter.

S is DOUBLE PRECISION

S is an array, dimension (N). If INFO = 0, S contains the scale factors for A.

SCOND Output parameter.

SCOND is DOUBLE PRECISION

If INFO = 0, S contains the ratio of the smallest S(i) to the largest S(i). If SCOND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by S.

AMAX Output parameter.

AMAX is DOUBLE PRECISION

Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element is nonpositive.

Related Information

For this routine in other precisions, please see *cppequ*, *dppequ* and *sppequ*. It also exists with a native C interface as *LAPACKE_zppequ*.

4.7.38 zsyequb

zsyequb computes row and column scalings intended to equilibrate a symmetric matrix A (with respect to the Euclidean norm) and reduce its condition number. The scale factors S are computed by the BIN algorithm (see references) so that the scaled matrix B with elements B(i,j) = S(i)*A(i,j)*S(j) has a condition number within a factor N of the smallest possible condition number over all possible diagonal scalings.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsyequb(UPLO, N, A, LDA, S, SCOND, AMAX, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zsyequb_(char *uplo, const armpl_int_t *n,
              const armpl_doublecomplex_t *a, const armpl_int_t *lda,
              double *s, double *scond, double *amax,
              armpl_doublecomplex_t *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The N-by-N symmetric matrix whose scaling factors are to be computed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

S Output parameter.

S is DOUBLE PRECISION

S is an array, dimension (N). If INFO = 0, S contains the scale factors for A.

SCOND Output parameter.

SCOND is DOUBLE PRECISION

If INFO = 0, S contains the ratio of the smallest S(i) to the largest S(i). If SCOND >= 0.1 and AMAX is neither too large nor too small, it is not worth scaling by S.

AMAX Output parameter.

AMAX is DOUBLE PRECISION

Largest absolute value of any matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element is nonpositive.

Related Information

For this routine in other precisions, please see *csyequb*, *dsyequb* and *ssyequb*. It also exists with a native C interface as *LAPACKE_zsyequb*.

4.8 LAPACK cosine sine routines

4.8.1 cbbcsd

cbbcsd computes the CS decomposition of a unitary matrix in bidiagonal-block form,

```
[ B11 | B12 0  0 ]
[  0  |  0 -I  0 ]
```

X = [————————-]

```
[ B21 | B22 0  0 ]
[  0  |  0  0  I ]

                           [  C | -S  0  0 ]
             [ U1 |    ] [  0 |  0 -I  0 ] [ V1 |    ]**H
           = [---------] [----------------] [---------]   .
             [    | U2 ] [  S |  C  0  0 ] [    | V2 ]
                           [  0 |  0  0  I ]
```

X is M-by-M, its top-left block is P-by-Q, and Q must be no larger than P, M-P, or M-Q. (If Q is not the smallest index, then X must be transposed and/or permuted. This can be done in constant time using the TRANS and SIGNS options. See CUNCSD for details.)

The bidiagonal matrices B11, B12, B21, and B22 are represented implicitly by angles THETA(1:Q) and PHI(1:Q-1).

The unitary matrices U1, U2, V1T, and V2T are input/output. The input matrices are pre- or post-multiplied by the appropriate singular vector matrices.

Syntax

Fortran specification:

```
use armpl_library

subroutine cbbcsd(JOBU1, JOBU2, JOBV1T, JOBV2T, TRANS, M, P, Q, THETA, PHI,
                  U1, LDU1, U2, LDU2, V1T, LDV1T, V2T, LDV2T, B11D, B11E,
                  B12D, B12E, B21D, B21E, B22D, B22E, RWORK, LRWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cbbcsd_(const char *jobu1, const char *jobu2, const char *jobv1t,
             const char *jobv2t, const char *trans, const armpl_int_t *m,
             const armpl_int_t *p, const armpl_int_t *q, float *theta,
```

(continues on next page)

(continued from previous page)

```
                float *phi, armpl_singlecomplex_t *u1, const armpl_int_t *ldu1,
                armpl_singlecomplex_t *u2, const armpl_int_t *ldu2,
                armpl_singlecomplex_t *v1t, const armpl_int_t *ldv1t,
                armpl_singlecomplex_t *v2t, const armpl_int_t *ldv2t,
                float *b11d, float *b11e, float *b12d, float *b12e, float *b21d,
                float *b21e, float *b22d, float *b22e, float *rwork,
                const armpl_int_t *lrwork, armpl_int_t *info, ... );
```

Parameters

JOBU1 Input parameter.

JOBU1 is CHARACTER

= 'Y': U1 is updated; otherwise: U1 is not updated.

JOBU2 Input parameter.

JOBU2 is CHARACTER

= 'Y': U2 is updated; otherwise: U2 is not updated.

JOBV1T Input parameter.

JOBV1T is CHARACTER

= 'Y': V1T is updated; otherwise: V1T is not updated.

JOBV2T Input parameter.

JOBV2T is CHARACTER

= 'Y': V2T is updated; otherwise: V2T is not updated.

TRANS Input parameter.

TRANS is CHARACTER

= 'T': X, U1, U2, V1T, and V2T are stored in row-major order; otherwise: X, U1, U2, V1T, and V2T are stored in column- major order.

M Input parameter.

M is INTEGER

The number of rows and columns in X, the unitary matrix in bidiagonal-block form.

P Input parameter.

P is INTEGER

The number of rows in the top-left block of X. 0 <= P <= M.

Q Input parameter.

Q is INTEGER

The number of columns in the top-left block of X. 0 <= Q <= MIN(P,M-P,M-Q).

THETA Input and output parameter.

THETA is REAL

THETA is an array, dimension (Q). On entry, the angles THETA(1),...,THETA(Q) that, along with PHI(1), ...,PHI(Q-1), define the matrix in bidiagonal-block form. On exit, the angles whose cosines and sines define the diagonal blocks in the CS decomposition.

PHI Input and output parameter.

PHI is REAL

PHI is an array, dimension (Q-1). The angles PHI(1),...,PHI(Q-1) that, along with THETA(1),..., THETA(Q), define the matrix in bidiagonal-block form.

U1 Input and output parameter.

U1 is COMPLEX

U1 is an array, dimension (LDU1, P). On entry, a P-by-P matrix. On exit, U1 is postmultiplied by the left singular vector matrix common to [B11 ; 0] and [B12 0 0 ; 0 -I 0 0].

LDU1 Input parameter.

LDU1 is INTEGER

The leading dimension of the array U1, LDU1 >= MAX(1, P).

U2 Input and output parameter.

U2 is COMPLEX

U2 is an array, dimension (LDU2,M-P). On entry, an (M-P)-by-(M-P) matrix. On exit, U2 is postmultiplied by the left singular vector matrix common to [B21 ; 0] and [B22 0 0 ; 0 0 I].

LDU2 Input parameter.

LDU2 is INTEGER

The leading dimension of the array U2, LDU2 >= MAX(1,M-P).

V1T Input and output parameter.

V1T is COMPLEX

V1T is an array, dimension (LDV1T, Q). On entry, a Q-by-Q matrix. On exit, V1T is premultiplied by the conjugate transpose of the right singular vector matrix common to [B11 ; 0] and [B21 ; 0].

LDV1T Input parameter.

LDV1T is INTEGER

The leading dimension of the array V1T, LDV1T >= MAX(1, Q).

V2T Input and output parameter.

V2T is COMPLEX

V2T is an array, dimension (LDV2T,M-Q). On entry, an (M-Q)-by-(M-Q) matrix. On exit, V2T is premultiplied by the conjugate transpose of the right singular vector matrix common to [B12 0 0 ; 0 -I 0] and [B22 0 0 ; 0 0 I].

LDV2T Input parameter.

LDV2T is INTEGER

The leading dimension of the array V2T, LDV2T >= MAX(1,M-Q).

B11D Output parameter.

B11D is REAL

B11D is an array, dimension (Q). When CBBCSD converges, B11D contains the cosines of THETA(1), ..., THETA(Q). If CBBCSD fails to converge, then B11D contains the diagonal of the partially reduced top-left block.

B11E Output parameter.

B11E is REAL

B11E is an array, dimension (Q-1). When CBBCSD converges, B11E contains zeros. If CBBCSD fails to converge, then B11E contains the superdiagonal of the partially reduced top-left block.

B12D Output parameter.

B12D is REAL

B12D is an array, dimension (Q). When CBBCSD converges, B12D contains the negative sines of THETA(1), ..., THETA(Q). If CBBCSD fails to converge, then B12D contains the diagonal of the partially reduced top-right block.

B12E Output parameter.

B12E is REAL

B12E is an array, dimension (Q-1). When CBBCSD converges, B12E contains zeros. If CBBCSD fails to converge, then B12E contains the subdiagonal of the partially reduced top-right block.

B21D Output parameter.

B21D is REAL

B21D is an array, dimension (Q). When CBBCSD converges, B21D contains the negative sines of THETA(1), ..., THETA(Q). If CBBCSD fails to converge, then B21D contains the diagonal of the partially reduced bottom-left block.

B21E Output parameter.

B21E is REAL

B21E is an array, dimension (Q-1). When CBBCSD converges, B21E contains zeros. If CBBCSD fails to converge, then B21E contains the subdiagonal of the partially reduced bottom-left block.

B22D Output parameter.

B22D is REAL

B22D is an array, dimension (Q). When CBBCSD converges, B22D contains the negative sines of THETA(1), ..., THETA(Q). If CBBCSD fails to converge, then B22D contains the diagonal of the partially reduced bottom-right block.

B22E Output parameter.

B22E is REAL

B22E is an array, dimension (Q-1). When CBBCSD converges, B22E contains zeros. If CBBCSD fails to converge, then B22E contains the subdiagonal of the partially reduced bottom-right block.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (MAX(1, LRWORK)). On exit, if INFO = 0, RWORK(1) returns the optimal LRWORK.

LRWORK Input parameter.

LRWORK is INTEGER

The dimension of the array RWORK. LRWORK >= MAX(1,8*Q).

If LRWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the RWORK array, returns this value as the first entry of the work array, and no error message related to LRWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if CBBCSD did not converge, INFO specifies the number of nonzero entries in PHI, and B11D, B11E, etc., contain the partially reduced matrix.

Related Information

For this routine in other precisions, please see *dbbcsd*, *sbbcsd* and *zbbcsd*. It also exists with a native C interface as *LAPACKE_cbbcsd*.

4.8.2 cunbdb

cunbdb simultaneously bidiagonalizes the blocks of an M-by-M partitioned unitary matrix X:

```
                                [ B11 | B12 0  0 ]
[ X11 | X12 ]   [ P1 |    ] [  0  |  0 -I  0 ] [ Q1 |    ]**H
```

X = [————] = [————] [————————-] [————] .

```
[ X21 | X22 ]   [    | P2 ] [ B21 | B22 0  0 ] [    | Q2 ]
                                [  0  |  0  0  I ]
```

X11 is P-by-Q. Q must be no larger than P, M-P, or M-Q. (If this is not the case, then X must be transposed and/or permuted. This can be done in constant time using the TRANS and SIGNS options. See CUNCSD for details.)

The unitary matrices P1, P2, Q1, and Q2 are P-by-P, (M-P)-by- (M-P), Q-by-Q, and (M-Q)-by-(M-Q), respectively. They are represented implicitly by Householder vectors.

B11, B12, B21, and B22 are Q-by-Q bidiagonal matrices represented implicitly by angles THETA, PHI.

Syntax

Fortran specification:

```
use armpl_library

subroutine cunbdb(TRANS, SIGNS, M, P, Q, X11, LDX11, X12, LDX12, X21, LDX21,
                  X22, LDX22, THETA, PHI, TAUP1, TAUP2, TAUQ1, TAUQ2, WORK,
                  LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cunbdb_(const char *trans, const char *signs, const armpl_int_t *m,
             const armpl_int_t *p, const armpl_int_t *q,
             armpl_singlecomplex_t *x11, const armpl_int_t *ldx11,
             armpl_singlecomplex_t *x12, const armpl_int_t *ldx12,
             armpl_singlecomplex_t *x21, const armpl_int_t *ldx21,
             armpl_singlecomplex_t *x22, const armpl_int_t *ldx22,
             float *theta, float *phi, armpl_singlecomplex_t *taup1,
             armpl_singlecomplex_t *taup2, armpl_singlecomplex_t *tauq1,
             armpl_singlecomplex_t *tauq2, armpl_singlecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER

= ‘T’: X, U1, U2, V1T, and V2T are stored in row-major order; otherwise: X, U1, U2, V1T, and V2T are stored in column- major order.

SIGNS Input parameter.

SIGNS is CHARACTER

= 'O': The lower-left block is made nonpositive (the "other" convention); otherwise: The upper-right block is made nonpositive (the "default" convention).

M Input parameter.

M is INTEGER

The number of rows and columns in X.

P Input parameter.

P is INTEGER

The number of rows in X11 and X12. 0 <= P <= M.

Q Input parameter.

Q is INTEGER

The number of columns in X11 and X21. 0 <= Q <= MIN(P,M-P,M-Q).

X11 Input and output parameter.

X11 is COMPLEX

X11 is an array, dimension (LDX11, Q). On entry, the top-left block of the unitary matrix to be reduced. On exit, the form depends on TRANS: If TRANS = 'N', then the columns of tril(X11) specify reflectors for P1, the rows of triu(X11,1) specify reflectors for Q1; else TRANS = 'T', and the rows of triu(X11) specify reflectors for P1, the columns of tril(X11,-1) specify reflectors for Q1.

LDX11 Input parameter.

LDX11 is INTEGER

The leading dimension of X11. If TRANS = 'N', then LDX11 >= P; else LDX11 >= Q.

X12 Input and output parameter.

X12 is COMPLEX

X12 is an array, dimension (LDX12,M-Q). On entry, the top-right block of the unitary matrix to be reduced. On exit, the form depends on TRANS: If TRANS = 'N', then the rows of triu(X12) specify the first P reflectors for Q2; else TRANS = 'T', and the columns of tril(X12) specify the first P reflectors for Q2.

LDX12 Input parameter.

LDX12 is INTEGER

The leading dimension of X12. If TRANS = 'N', then LDX12 >= P; else LDX11 >= M-Q.

X21 Input and output parameter.

X21 is COMPLEX

X21 is an array, dimension (LDX21, Q). On entry, the bottom-left block of the unitary matrix to be reduced. On exit, the form depends on TRANS: If TRANS = 'N', then the columns of tril(X21) specify reflectors for P2; else TRANS = 'T', and the rows of triu(X21) specify reflectors for P2.

LDX21 Input parameter.

LDX21 is INTEGER

The leading dimension of X21. If TRANS = 'N', then LDX21 >= M-P; else LDX21 >= Q.

X22 Input and output parameter.

X22 is COMPLEX

X22 is an array, dimension (LDX22,M-Q). On entry, the bottom-right block of the unitary matrix to be reduced. On exit, the form depends on TRANS: If TRANS = 'N', then the rows of triu(X22(Q+1:M-P,P+1:M-Q)) specify the last M-P-Q reflectors for Q2, else TRANS = 'T', and the columns of tril(X22(P+1:M-Q,Q+1:M-P)) specify the last M-P-Q reflectors for P2.

LDX22 Input parameter.

LDX22 is INTEGER

The leading dimension of X22. If TRANS = 'N', then LDX22 >= M-P; else LDX22 >= M-Q.

THETA Output parameter.

THETA is REAL

THETA is an array, dimension (Q). The entries of the bidiagonal blocks B11, B12, B21, B22 can be computed from the angles THETA and PHI. See Further Details.

PHI Output parameter.

PHI is REAL

PHI is an array, dimension (Q-1). The entries of the bidiagonal blocks B11, B12, B21, B22 can be computed from the angles THETA and PHI. See Further Details.

TAUP1 Output parameter.

TAUP1 is COMPLEX

TAUP1 is an array, dimension (P). The scalar factors of the elementary reflectors that define P1.

TAUP2 Output parameter.

TAUP2 is COMPLEX

TAUP2 is an array, dimension (M-P). The scalar factors of the elementary reflectors that define P2.

TAUQ1 Output parameter.

TAUQ1 is COMPLEX

TAUQ1 is an array, dimension (Q). The scalar factors of the elementary reflectors that define Q1.

TAUQ2 Output parameter.

TAUQ2 is COMPLEX

TAUQ2 is an array, dimension (M-Q). The scalar factors of the elementary reflectors that define Q2.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= M-Q.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *zunbdb*. It also exists with a native C interface as *LAPACKE_cunbdb*.

4.8.3 cuncsd

`cuncsd` computes the CS decomposition of an M-by-M partitioned unitary matrix X:

```
                                [  I  0  0 |  0  0  0 ]
                                [  0  C  0 |  0 -S  0 ]
[ X11 | X12 ]   [ U1 |    ] [  0  0  0 |  0  0 -I ] [ V1 |    ]**H
```

X = [————] = [———] [————————] [———] .

```
[ X21 | X22 ]   [    | U2 ] [  0  0  0 |  I  0  0 ] [    | V2 ]
                                [  0  S  0 |  0  C  0 ]
                                [  0  0  I |  0  0  0 ]
```

X11 is P-by-Q. The unitary matrices U1, U2, V1, and V2 are P-by-P, (M-P)-by-(M-P), Q-by-Q, and (M-Q)-by-(M-Q), respectively. C and S are R-by-R nonnegative diagonal matrices satisfying C^2 + S^2 = I, in which R = MIN(P,M-P,Q,M-Q).

Syntax

Fortran specification:

```
use armpl_library

recursive subroutine cuncsd(JOBU1, JOBU2, JOBV1T, JOBV2T, TRANS, SIGNS, M, P,
                            Q, X11, LDX11, X12, LDX12, X21, LDX21, X22, LDX22,
                            THETA, U1, LDU1, U2, LDU2, V1T, LDV1T, V2T, LDV2T,
                            WORK, LWORK, RWORK, LRWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cuncsd_(const char *jobu1, const char *jobu2, const char *jobv1t,
             const char *jobv2t, const char *trans, const char *signs,
             const armpl_int_t *m, const armpl_int_t *p, const armpl_int_t *q,
             armpl_singlecomplex_t *x11, const armpl_int_t *ldx11,
             armpl_singlecomplex_t *x12, const armpl_int_t *ldx12,
             armpl_singlecomplex_t *x21, const armpl_int_t *ldx21,
             armpl_singlecomplex_t *x22, const armpl_int_t *ldx22,
             float *theta, armpl_singlecomplex_t *u1, const armpl_int_t *ldu1,
             armpl_singlecomplex_t *u2, const armpl_int_t *ldu2,
             armpl_singlecomplex_t *v1t, const armpl_int_t *ldv1t,
             armpl_singlecomplex_t *v2t, const armpl_int_t *ldv2t,
             armpl_singlecomplex_t *work, const armpl_int_t *lwork,
             float *rwork, const armpl_int_t *lrwork, armpl_int_t *iwork,
             armpl_int_t *info, ... );
```

Parameters

JOBU1 Input parameter.

`JOBU1` is CHARACTER

= 'Y': `U1` is computed; otherwise: `U1` is not computed.

JOBU2 Input parameter.

JOBU2 is CHARACTER

= 'Y': U2 is computed; otherwise: U2 is not computed.

JOBV1T Input parameter.

JOBV1T is CHARACTER

= 'Y': V1T is computed; otherwise: V1T is not computed.

JOBV2T Input parameter.

JOBV2T is CHARACTER

= 'Y': V2T is computed; otherwise: V2T is not computed.

TRANS Input parameter.

TRANS is CHARACTER

= 'T': X, U1, U2, V1T, and V2T are stored in row-major order; otherwise: X, U1, U2, V1T, and V2T are stored in column- major order.

SIGNS Input parameter.

SIGNS is CHARACTER

= 'O': The lower-left block is made nonpositive (the "other" convention); otherwise: The upper-right block is made nonpositive (the "default" convention).

M Input parameter.

M is INTEGER

The number of rows and columns in X.

P Input parameter.

P is INTEGER

The number of rows in X11 and X12. 0 <= P <= M.

Q Input parameter.

Q is INTEGER

The number of columns in X11 and X21. 0 <= Q <= M.

X11 Input and output parameter.

X11 is COMPLEX

X11 is an array, dimension (LDX11, Q). On entry, part of the unitary matrix whose CSD is desired.

LDX11 Input parameter.

LDX11 is INTEGER

The leading dimension of X11. LDX11 >= MAX(1, P).

X12 Input and output parameter.

X12 is COMPLEX

X12 is an array, dimension (LDX12,M-Q). On entry, part of the unitary matrix whose CSD is desired.

LDX12 Input parameter.

LDX12 is INTEGER

The leading dimension of X12. LDX12 >= MAX(1, P).

X21 Input and output parameter.

X21 is COMPLEX

X21 is an array, dimension (LDX21, Q). On entry, part of the unitary matrix whose CSD is desired.

LDX21 Input parameter.

LDX21 is INTEGER

The leading dimension of X11. LDX21 >= MAX(1,M-P).

X22 Input and output parameter.

X22 is COMPLEX

X22 is an array, dimension (LDX22,M-Q). On entry, part of the unitary matrix whose CSD is desired.

LDX22 Input parameter.

LDX22 is INTEGER

The leading dimension of X11. LDX22 >= MAX(1,M-P).

THETA Output parameter.

THETA is REAL

THETA is an array, dimension (R), in which R =. MIN(P,M-P, Q,M-Q). C = DIAG(COS(THETA(1)), ... , COS(THETA(R))) and S = DIAG(SIN(THETA(1)), ... , SIN(THETA(R))).

U1 Output parameter.

U1 is COMPLEX

U1 is an array, dimension (LDU1, P). If JOBU1 = 'Y', U1 contains the P-by-P unitary matrix U1.

LDU1 Input parameter.

LDU1 is INTEGER

The leading dimension of U1. If JOBU1 = 'Y', LDU1 >= MAX(1, P).

U2 Output parameter.

U2 is COMPLEX

U2 is an array, dimension (LDU2,M-P). If JOBU2 = 'Y', U2 contains the (M-P)-by-(M-P) unitary matrix U2.

LDU2 Input parameter.

LDU2 is INTEGER

The leading dimension of U2. If JOBU2 = 'Y', LDU2 >= MAX(1,M-P).

V1T Output parameter.

V1T is COMPLEX

V1T is an array, dimension (LDV1T, Q). If JOBV1T = 'Y', V1T contains the Q-by-Q matrix unitary matrix $V1^H$.

LDV1T Input parameter.

LDV1T is INTEGER

The leading dimension of V1T. If JOBV1T = 'Y', LDV1T >= MAX(1, Q).

V2T Output parameter.

V2T is COMPLEX

V2T is an array, dimension (LDV2T,M-Q). If JOBV2T = 'Y', V2T contains the (M-Q)-by-(M-Q) unitary matrix $V2^H$.

LDV2T Input parameter.

LDV2T is INTEGER

The leading dimension of V2T. If JOBV2T = 'Y', LDV2T >= MAX(1,M-Q).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the work array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension MAX(1, LRWORK). On exit, if INFO = 0, RWORK(1) returns the optimal LRWORK. If INFO > 0 on exit, RWORK(2:R) contains the values PHI(1), ..., PHI(R-1) that, together with THETA(1), ..., THETA(R), define the matrix in intermediate bidiagonal-block form remaining after nonconvergence. INFO specifies the number of nonzero PHI's.

LRWORK Input parameter.

LRWORK is INTEGER

The dimension of the array RWORK.

If LRWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the RWORK array, returns this value as the first entry of the work array, and no error message related to LRWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (M-MIN(P,M-P, Q,M-Q))

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: CBBCSD did not converge. See the description of RWORK above for details.

Related Information

For this routine in other precisions, please see *zuncsd*. It also exists with a native C interface as *LAPACKE_cuncsd*.

4.8.4 cuncsd2by1

CUNCSD2BY1 computes the CS decomposition of an M-by-Q matrix X with orthonormal columns that has been partitioned into a 2-by-1 block structure:

```
                           [  I1 0  0 ]
                           [  0  C  0 ]
    [ X11 ]   [ U1 |    ] [  0  0  0 ]
X = [-----] = [---------] [----------] V1**T .
    [ X21 ]   [    | U2 ] [  0  0  0 ]
```

(continues on next page)

(continued from previous page)

```
                          [  0   S   0 ]
                          [  0   0   I2]
```

X11 is P-by-Q. The unitary matrices U1, U2, and V1 are P-by-P, (M-P)-by-(M-P), and Q-by-Q, respectively. C and S are R-by-R nonnegative diagonal matrices satisfying C^2 + S^2 = I, in which R = MIN(P,M-P,Q,M-Q). I1 is a K1-by-K1 identity matrix and I2 is a K2-by-K2 identity matrix, where K1 = MAX(Q+P-M,0), K2 = MAX(Q-P,0).

Syntax

Fortran specification:

```
use armpl_library

subroutine cuncsd2by1(JOBU1, JOBU2, JOBV1T, M, P, Q, X11, LDX11, X21, LDX21,
                      THETA, U1, LDU1, U2, LDU2, V1T, LDV1T, WORK, LWORK,
                      RWORK, LRWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cuncsd2by1_(const char *jobu1, const char *jobu2, const char *jobv1t,
                 const armpl_int_t *m, const armpl_int_t *p,
                 const armpl_int_t *q, armpl_singlecomplex_t *x11,
                 const armpl_int_t *ldx11, armpl_singlecomplex_t *x21,
                 const armpl_int_t *ldx21, float *theta,
                 armpl_singlecomplex_t *u1, const armpl_int_t *ldu1,
                 armpl_singlecomplex_t *u2, const armpl_int_t *ldu2,
                 armpl_singlecomplex_t *v1t, const armpl_int_t *ldv1t,
                 armpl_singlecomplex_t *work, const armpl_int_t *lwork,
                 float *rwork, const armpl_int_t *lrwork, armpl_int_t *iwork,
                 armpl_int_t *info, ... );
```

Parameters

JOBU1 Input parameter.

JOBU1 is CHARACTER

= 'Y': U1 is computed; otherwise: U1 is not computed.

JOBU2 Input parameter.

JOBU2 is CHARACTER

= 'Y': U2 is computed; otherwise: U2 is not computed.

JOBV1T Input parameter.

JOBV1T is CHARACTER

= 'Y': V1T is computed; otherwise: V1T is not computed.

M Input parameter.

M is INTEGER

The number of rows in X.

P Input parameter.

P is INTEGER

The number of rows in X11. 0 <= P <= M.

Q Input parameter.

Q is INTEGER

The number of columns in X11 and X21. 0 <= Q <= M.

X11 Input and output parameter.

X11 is COMPLEX

X11 is an array, dimension (LDX11, Q). On entry, part of the unitary matrix whose CSD is desired.

LDX11 Input parameter.

LDX11 is INTEGER

The leading dimension of X11. LDX11 >= MAX(1, P).

X21 Input and output parameter.

X21 is COMPLEX

X21 is an array, dimension (LDX21, Q). On entry, part of the unitary matrix whose CSD is desired.

LDX21 Input parameter.

LDX21 is INTEGER

The leading dimension of X21. LDX21 >= MAX(1,M-P).

THETA Output parameter.

THETA is REAL

THETA is an array, dimension (R), in which R =. MIN(P,M-P, Q,M-Q). C = DIAG(COS(THETA(1)), ... , COS(THETA(R))) and S = DIAG(SIN(THETA(1)), ... , SIN(THETA(R))).

U1 Output parameter.

U1 is COMPLEX

U1 is an array, dimension (P). If JOBU1 = 'Y', U1 contains the P-by-P unitary matrix U1.

LDU1 Input parameter.

LDU1 is INTEGER

The leading dimension of U1. If JOBU1 = 'Y', LDU1 >= MAX(1, P).

U2 Output parameter.

U2 is COMPLEX

U2 is an array, dimension (M-P). If JOBU2 = 'Y', U2 contains the (M-P)-by-(M-P) unitary matrix U2.

LDU2 Input parameter.

LDU2 is INTEGER

The leading dimension of U2. If JOBU2 = 'Y', LDU2 >= MAX(1,M-P).

V1T Output parameter.

V1T is COMPLEX

V1T is an array, dimension (Q). If JOBV1T = 'Y', V1T contains the Q-by-Q matrix unitary matrix $V1^T$.

LDV1T Input parameter.

LDV1T is INTEGER

The leading dimension of V1T. If JOBV1T = 'Y', LDV1T >= MAX(1, Q).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the work array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (MAX(1, LRWORK)). On exit, if INFO = 0, RWORK(1) returns the optimal LRWORK. If INFO > 0 on exit, RWORK(2:R) contains the values PHI(1), ..., PHI(R-1) that, together with THETA(1), ..., THETA(R), define the matrix in intermediate bidiagonal-block form remaining after nonconvergence. INFO specifies the number of nonzero PHI's.

LRWORK Input parameter.

LRWORK is INTEGER

The dimension of the array RWORK.

If LRWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the RWORK array, returns this value as the first entry of the work array, and no error message related to LRWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (M-MIN(P,M-P, Q,M-Q))

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: CBBCSD did not converge. See the description of WORK above for details.

Related Information

For this routine in other precisions, please see *zuncsd2by1*. It also exists with a native C interface as *LAPACKE_cuncsd2by1*.

4.8.5 dbbcsd

dbbcsd computes the CS decomposition of an orthogonal matrix in bidiagonal-block form,

```
[ B11 | B12 0  0 ]
[  0  |  0 -I  0 ]
```

X = [————————-]

```
[ B21 | B22 0  0 ]
[  0  |  0  0  I ]

                          [  C | -S  0  0 ]
              [ U1 |    ] [  0 |  0 -I  0 ] [ V1 |    ]**T
```

(continues on next page)

(continued from previous page)

```
             = [---------] [---------------] [---------]    .
               [     | U2 ] [  S |  C   0   0 ] [     | V2 ]
                            [  0 |  0   0   I ]
```

X is M-by-M, its top-left block is P-by-Q, and Q must be no larger than P, M-P, or M-Q. (If Q is not the smallest index, then X must be transposed and/or permuted. This can be done in constant time using the TRANS and SIGNS options. See DORCSD for details.)

The bidiagonal matrices B11, B12, B21, and B22 are represented implicitly by angles THETA(1:Q) and PHI(1:Q-1).

The orthogonal matrices U1, U2, V1T, and V2T are input/output. The input matrices are pre- or post-multiplied by the appropriate singular vector matrices.

Syntax

Fortran specification:

```
use armpl_library

subroutine dbbcsd(JOBU1, JOBU2, JOBV1T, JOBV2T, TRANS, M, P, Q, THETA, PHI,
                  U1, LDU1, U2, LDU2, V1T, LDV1T, V2T, LDV2T, B11D, B11E,
                  B12D, B12E, B21D, B21E, B22D, B22E, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dbbcsd_(const char *jobu1, const char *jobu2, const char *jobv1t,
             const char *jobv2t, const char *trans, const armpl_int_t *m,
             const armpl_int_t *p, const armpl_int_t *q, double *theta,
             double *phi, double *u1, const armpl_int_t *ldu1, double *u2,
             const armpl_int_t *ldu2, double *v1t, const armpl_int_t *ldv1t,
             double *v2t, const armpl_int_t *ldv2t, double *b11d,
             double *b11e, double *b12d, double *b12e, double *b21d,
             double *b21e, double *b22d, double *b22e, double *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

JOBU1 Input parameter.

JOBU1 is CHARACTER

= 'Y': U1 is updated; otherwise: U1 is not updated.

JOBU2 Input parameter.

JOBU2 is CHARACTER

= 'Y': U2 is updated; otherwise: U2 is not updated.

JOBV1T Input parameter.

JOBV1T is CHARACTER

= 'Y': V1T is updated; otherwise: V1T is not updated.

JOBV2T Input parameter.

JOBV2T is CHARACTER

= 'Y': V2T is updated; otherwise: V2T is not updated.

TRANS Input parameter.

TRANS is CHARACTER

= 'T': X, U1, U2, V1T, and V2T are stored in row-major order; otherwise: X, U1, U2, V1T, and V2T are stored in column- major order.

M Input parameter.

M is INTEGER

The number of rows and columns in X, the orthogonal matrix in bidiagonal-block form.

P Input parameter.

P is INTEGER

The number of rows in the top-left block of X. 0 <= P <= M.

Q Input parameter.

Q is INTEGER

The number of columns in the top-left block of X. 0 <= Q <= MIN(P,M-P,M-Q).

THETA Input and output parameter.

THETA is DOUBLE PRECISION

THETA is an array, dimension (Q). On entry, the angles THETA(1),...,THETA(Q) that, along with PHI(1), ...,PHI(Q-1), define the matrix in bidiagonal-block form. On exit, the angles whose cosines and sines define the diagonal blocks in the CS decomposition.

PHI Input and output parameter.

PHI is DOUBLE PRECISION

PHI is an array, dimension (Q-1). The angles PHI(1),...,PHI(Q-1) that, along with THETA(1),..., THETA(Q), define the matrix in bidiagonal-block form.

U1 Input and output parameter.

U1 is DOUBLE PRECISION

U1 is an array, dimension (LDU1, P). On entry, a P-by-P matrix. On exit, U1 is postmultiplied by the left singular vector matrix common to [B11 ; 0] and [B12 0 0 ; 0 -I 0 0].

LDU1 Input parameter.

LDU1 is INTEGER

The leading dimension of the array U1, LDU1 >= MAX(1, P).

U2 Input and output parameter.

U2 is DOUBLE PRECISION

U2 is an array, dimension (LDU2,M-P). On entry, an (M-P)-by-(M-P) matrix. On exit, U2 is postmultiplied by the left singular vector matrix common to [B21 ; 0] and [B22 0 0 ; 0 0 I].

LDU2 Input parameter.

LDU2 is INTEGER

The leading dimension of the array U2, LDU2 >= MAX(1,M-P).

V1T Input and output parameter.

V1T is DOUBLE PRECISION

V1T is an array, dimension (LDV1T, Q). On entry, a Q-by-Q matrix. On exit, V1T is premultiplied by the transpose of the right singular vector matrix common to [B11 ; 0] and [B21 ; 0].

LDV1T Input parameter.

LDV1T is INTEGER

The leading dimension of the array V1T, LDV1T >= MAX(1, Q).

V2T Input and output parameter.

V2T is DOUBLE PRECISION

V2T is an array, dimension (LDV2T,M-Q). On entry, an (M-Q)-by-(M-Q) matrix. On exit, V2T is premultiplied by the transpose of the right singular vector matrix common to [B12 0 0 ; 0 -I 0] and [B22 0 0 ; 0 0 I].

LDV2T Input parameter.

LDV2T is INTEGER

The leading dimension of the array V2T, LDV2T >= MAX(1,M-Q).

B11D Output parameter.

B11D is DOUBLE PRECISION

B11D is an array, dimension (Q). When DBBCSD converges, B11D contains the cosines of THETA(1), ..., THETA(Q). If DBBCSD fails to converge, then B11D contains the diagonal of the partially reduced top-left block.

B11E Output parameter.

B11E is DOUBLE PRECISION

B11E is an array, dimension (Q-1). When DBBCSD converges, B11E contains zeros. If DBBCSD fails to converge, then B11E contains the superdiagonal of the partially reduced top-left block.

B12D Output parameter.

B12D is DOUBLE PRECISION

B12D is an array, dimension (Q). When DBBCSD converges, B12D contains the negative sines of THETA(1), ..., THETA(Q). If DBBCSD fails to converge, then B12D contains the diagonal of the partially reduced top-right block.

B12E Output parameter.

B12E is DOUBLE PRECISION

B12E is an array, dimension (Q-1). When DBBCSD converges, B12E contains zeros. If DBBCSD fails to converge, then B12E contains the subdiagonal of the partially reduced top-right block.

B21D Output parameter.

B21D is DOUBLE PRECISION array, dimension (Q)

When DBBCSD converges, B21D contains the negative sines of THETA(1), ..., THETA(Q). If DBBCSD fails to converge, then B21D contains the diagonal of the partially reduced bottom-left block.

B21E Output parameter.

B21E is DOUBLE PRECISION array, dimension (Q-1)

When DBBCSD converges, B21E contains zeros. If DBBCSD fails to converge, then B21E contains the subdiagonal of the partially reduced bottom-left block.

B22D Output parameter.

B22D is DOUBLE PRECISION array, dimension (Q)

When DBBCSD converges, B22D contains the negative sines of THETA(1), ..., THETA(Q). If DBBCSD fails to converge, then B22D contains the diagonal of the partially reduced bottom-right block.

B22E Output parameter.

B22E is DOUBLE PRECISION array, dimension (Q-1)

When DBBCSD converges, B22E contains zeros. If DBBCSD fails to converge, then B22E contains the subdiagonal of the partially reduced bottom-right block.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= MAX(1,8*Q).

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the work array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if DBBCSD did not converge, INFO specifies the number of nonzero entries in PHI, and B11D, B11E, etc., contain the partially reduced matrix.

Related Information

For this routine in other precisions, please see *cbbcsd*, *sbbcsd* and *zbbcsd*. It also exists with a native C interface as *LAPACKE_dbbcsd*.

4.8.6 dorbdb

dorbdb simultaneously bidiagonalizes the blocks of an M-by-M partitioned orthogonal matrix X:

```
                                [ B11 | B12 0  0 ]
[ X11 | X12 ]   [ P1 |    ] [  0  |  0 -I  0 ] [ Q1 |    ]**T
```

X = [————] = [————] [————————-] [————] .

```
[ X21 | X22 ]   [    | P2 ] [ B21 | B22 0  0 ] [    | Q2 ]
                            [  0  |  0  0  I ]
```

X11 is P-by-Q. Q must be no larger than P, M-P, or M-Q. (If this is not the case, then X must be transposed and/or permuted. This can be done in constant time using the TRANS and SIGNS options. See DORCSD for details.)

The orthogonal matrices P1, P2, Q1, and Q2 are P-by-P, (M-P)-by- (M-P), Q-by-Q, and (M-Q)-by-(M-Q), respectively. They are represented implicitly by Householder vectors.

B11, B12, B21, and B22 are Q-by-Q bidiagonal matrices represented implicitly by angles THETA, PHI.

Syntax

Fortran specification:

```
use armpl_library

subroutine dorbdb(TRANS, SIGNS, M, P, Q, X11, LDX11, X12, LDX12, X21, LDX21,
                  X22, LDX22, THETA, PHI, TAUP1, TAUP2, TAUQ1, TAUQ2, WORK,
                  LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dorbdb_(const char *trans, const char *signs, const armpl_int_t *m,
             const armpl_int_t *p, const armpl_int_t *q, double *x11,
             const armpl_int_t *ldx11, double *x12, const armpl_int_t *ldx12,
             double *x21, const armpl_int_t *ldx21, double *x22,
             const armpl_int_t *ldx22, double *theta, double *phi,
             double *taup1, double *taup2, double *tauq1, double *tauq2,
             double *work, const armpl_int_t *lwork, armpl_int_t *info,
             ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER

= 'T': X, U1, U2, V1T, and V2T are stored in row-major order; otherwise: X, U1, U2, V1T, and V2T are stored in column- major order.

SIGNS Input parameter.

SIGNS is CHARACTER

= 'O': The lower-left block is made nonpositive (the "other" convention); otherwise: The upper-right block is made nonpositive (the "default" convention).

M Input parameter.

M is INTEGER

The number of rows and columns in X.

P Input parameter.

P is INTEGER

The number of rows in X11 and X12. 0 <= P <= M.

Q Input parameter.

Q is INTEGER

The number of columns in X11 and X21. 0 <= Q <= MIN(P,M-P,M-Q).

X11 Input and output parameter.

X11 is DOUBLE PRECISION

X11 is an array, dimension (LDX11, Q). On entry, the top-left block of the orthogonal matrix to be reduced. On exit, the form depends on TRANS: If TRANS = 'N', then the columns of tril(X11) specify reflectors for P1, the rows of triu(X11,1) specify reflectors for Q1; else TRANS = 'T', and the rows of triu(X11) specify reflectors for P1, the columns of tril(X11,-1) specify reflectors for Q1.

LDX11 Input parameter.

LDX11 is INTEGER

The leading dimension of X11. If TRANS = 'N', then LDX11 >= P; else LDX11 >= Q.

X12 Input and output parameter.

X12 is DOUBLE PRECISION

X12 is an array, dimension (LDX12,M-Q). On entry, the top-right block of the orthogonal matrix to be reduced. On exit, the form depends on TRANS: If TRANS = 'N', then the rows of triu(X12) specify the first P reflectors for Q2; else TRANS = 'T', and the columns of tril(X12) specify the first P reflectors for Q2.

LDX12 Input parameter.

LDX12 is INTEGER

The leading dimension of X12. If TRANS = 'N', then LDX12 >= P; else LDX11 >= M-Q.

X21 Input and output parameter.

X21 is DOUBLE PRECISION

X21 is an array, dimension (LDX21, Q). On entry, the bottom-left block of the orthogonal matrix to be reduced. On exit, the form depends on TRANS: If TRANS = 'N', then the columns of tril(X21) specify reflectors for P2; else TRANS = 'T', and the rows of triu(X21) specify reflectors for P2.

LDX21 Input parameter.

LDX21 is INTEGER

The leading dimension of X21. If TRANS = 'N', then LDX21 >= M-P; else LDX21 >= Q.

X22 Input and output parameter.

X22 is DOUBLE PRECISION

X22 is an array, dimension (LDX22,M-Q). On entry, the bottom-right block of the orthogonal matrix to be reduced. On exit, the form depends on TRANS: If TRANS = 'N', then the rows of triu(X22(Q+1:M-P,P+1:M-Q)) specify the last M-P-Q reflectors for Q2, else TRANS = 'T', and the columns of tril(X22(P+1:M-Q,Q+1:M-P)) specify the last M-P-Q reflectors for P2.

LDX22 Input parameter.

LDX22 is INTEGER

The leading dimension of X22. If TRANS = 'N', then LDX22 >= M-P; else LDX22 >= M-Q.

THETA Output parameter.

THETA is DOUBLE PRECISION

THETA is an array, dimension (Q). The entries of the bidiagonal blocks B11, B12, B21, B22 can be computed from the angles THETA and PHI. See Further Details.

PHI Output parameter.

PHI is DOUBLE PRECISION

PHI is an array, dimension (Q-1). The entries of the bidiagonal blocks B11, B12, B21, B22 can be computed from the angles THETA and PHI. See Further Details.

TAUP1 Output parameter.

TAUP1 is DOUBLE PRECISION

TAUP1 is an array, dimension (P). The scalar factors of the elementary reflectors that define P1.

TAUP2 Output parameter.

TAUP2 is DOUBLE PRECISION

TAUP2 is an array, dimension (M-P). The scalar factors of the elementary reflectors that define P2.

TAUQ1 Output parameter.

TAUQ1 is DOUBLE PRECISION

TAUQ1 is an array, dimension (Q). The scalar factors of the elementary reflectors that define Q1.

TAUQ2 Output parameter.

TAUQ2 is DOUBLE PRECISION

TAUQ2 is an array, dimension (M-Q). The scalar factors of the elementary reflectors that define Q2.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= M-Q.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *sorbdb*. It also exists with a native C interface as *LAPACKE_dorbdb*.

4.8.7 dorcsd

dorcsd computes the CS decomposition of an M-by-M partitioned orthogonal matrix X:

```
                              [  I  0  0 |  0  0  0 ]
                              [  0  C  0 |  0 -S  0 ]
[ X11 | X12 ]   [ U1 |    ] [  0  0  0 |  0  0 -I ] [ V1 |    ]**T
```

X = [————] = [———] [—————————] [———] .

```
[ X21 | X22 ]   [    | U2 ] [  0  0  0 |  I  0  0 ] [    | V2 ]
                              [  0  S  0 |  0  C  0 ]
                              [  0  0  I |  0  0  0 ]
```

X11 is P-by-Q. The orthogonal matrices U1, U2, V1, and V2 are P-by-P, (M-P)-by-(M-P), Q-by-Q, and (M-Q)-by-(M-Q), respectively. C and S are R-by-R nonnegative diagonal matrices satisfying C^2 + S^2 = I, in which R = MIN(P,M-P,Q,M-Q).

Syntax

Fortran specification:

```
use armpl_library

recursive subroutine dorcsd(JOBU1, JOBU2, JOBV1T, JOBV2T, TRANS, SIGNS, M, P,
                            Q, X11, LDX11, X12, LDX12, X21, LDX21, X22, LDX22,
                            THETA, U1, LDU1, U2, LDU2, V1T, LDV1T, V2T, LDV2T,
                            WORK, LWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dorcsd_(const char *jobu1, const char *jobu2, const char *jobv1t,
             const char *jobv2t, const char *trans, const char *signs,
             const armpl_int_t *m, const armpl_int_t *p, const armpl_int_t *q,
             double *x11, const armpl_int_t *ldx11, double *x12,
             const armpl_int_t *ldx12, double *x21, const armpl_int_t *ldx21,
             double *x22, const armpl_int_t *ldx22, double *theta, double *u1,
             const armpl_int_t *ldu1, double *u2, const armpl_int_t *ldu2,
             double *v1t, const armpl_int_t *ldv1t, double *v2t,
             const armpl_int_t *ldv2t, double *work, const armpl_int_t *lwork,
             armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

JOBU1 Input parameter.

JOBU1 is CHARACTER

= 'Y': U1 is computed; otherwise: U1 is not computed.

JOBU2 Input parameter.

JOBU2 is CHARACTER

= 'Y': U2 is computed; otherwise: U2 is not computed.

JOBV1T Input parameter.

JOBV1T is CHARACTER

= 'Y': V1T is computed; otherwise: V1T is not computed.

JOBV2T Input parameter.

JOBV2T is CHARACTER

= 'Y': V2T is computed; otherwise: V2T is not computed.

TRANS Input parameter.

TRANS is CHARACTER

= 'T': X, U1, U2, V1T, and V2T are stored in row-major order; otherwise: X, U1, U2, V1T, and V2T are stored in column- major order.

SIGNS Input parameter.

SIGNS is CHARACTER

= 'O': The lower-left block is made nonpositive (the "other" convention); otherwise: The upper-right block is made nonpositive (the "default" convention).

M Input parameter.

M is INTEGER

The number of rows and columns in X.

P Input parameter.

P is INTEGER

The number of rows in X11 and X12. 0 <= P <= M.

Q Input parameter.

Q is INTEGER

The number of columns in X11 and X21. 0 <= Q <= M.

X11 Input and output parameter.

X11 is DOUBLE PRECISION

X11 is an array, dimension (LDX11, Q). On entry, part of the orthogonal matrix whose CSD is desired.

LDX11 Input parameter.

LDX11 is INTEGER

The leading dimension of X11. LDX11 >= MAX(1, P).

X12 Input and output parameter.

X12 is DOUBLE PRECISION

X12 is an array, dimension (LDX12,M-Q). On entry, part of the orthogonal matrix whose CSD is desired.

LDX12 Input parameter.

LDX12 is INTEGER

The leading dimension of X12. LDX12 >= MAX(1, P).

X21 Input and output parameter.

X21 is DOUBLE PRECISION

X21 is an array, dimension (LDX21, Q). On entry, part of the orthogonal matrix whose CSD is desired.

LDX21 Input parameter.

LDX21 is INTEGER

The leading dimension of X11. LDX21 >= MAX(1,M-P).

X22 Input and output parameter.

X22 is DOUBLE PRECISION

X22 is an array, dimension (LDX22,M-Q). On entry, part of the orthogonal matrix whose CSD is desired.

LDX22 Input parameter.

LDX22 is INTEGER

The leading dimension of X11. LDX22 >= MAX(1,M-P).

THETA Output parameter.

THETA is DOUBLE PRECISION

THETA is an array, dimension (R), in which R =. MIN(P,M-P, Q,M-Q). C = DIAG(COS(THETA(1)), ... , COS(THETA(R))) and S = DIAG(SIN(THETA(1)), ... , SIN(THETA(R))).

U1 Output parameter.

U1 is DOUBLE PRECISION

U1 is an array, dimension (LDU1, P). If JOBU1 = 'Y', U1 contains the P-by-P orthogonal matrix U1.

LDU1 Input parameter.

LDU1 is INTEGER

The leading dimension of U1. If JOBU1 = 'Y', LDU1 >= MAX(1, P).

U2 Output parameter.

U2 is DOUBLE PRECISION

U2 is an array, dimension (LDU2,M-P). If JOBU2 = 'Y', U2 contains the (M-P)-by-(M-P) orthogonal matrix U2.

LDU2 Input parameter.

LDU2 is INTEGER

The leading dimension of U2. If JOBU2 = 'Y', LDU2 >= MAX(1,M-P).

V1T Output parameter.

V1T is DOUBLE PRECISION

V1T is an array, dimension (LDV1T, Q). If JOBV1T = 'Y', V1T contains the Q-by-Q matrix orthogonal matrix $V1^T$.

LDV1T Input parameter.

LDV1T is INTEGER

The leading dimension of V1T. If JOBV1T = 'Y', LDV1T >= MAX(1, Q).

V2T Output parameter.

V2T is DOUBLE PRECISION

V2T is an array, dimension (LDV2T,M-Q). If JOBV2T = 'Y', V2T contains the (M-Q)-by-(M-Q) orthogonal matrix $V2^T$.

LDV2T Input parameter.

LDV2T is INTEGER

The leading dimension of V2T. If JOBV2T = 'Y', LDV2T >= MAX(1,M-Q).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK. If INFO > 0 on exit, WORK(2:R) contains the values PHI(1), ..., PHI(R-1) that, together with THETA(1), ..., THETA(R), define the matrix in intermediate bidiagonal-block form remaining after nonconvergence. INFO specifies the number of nonzero PHI's.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the work array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (M-MIN(P, M-P, Q, M-Q))

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: DBBCSD did not converge. See the description of WORK above for details.

Related Information

For this routine in other precisions, please see *sorcsd*. It also exists with a native C interface as *LAPACKE_dorcsd*.

4.8.8 dorcsd2by1

DORCSD2BY1 computes the CS decomposition of an M-by-Q matrix X with orthonormal columns that has been partitioned into a 2-by-1 block structure:

```
                            [  I1 0  0 ]
                            [  0  C  0 ]
    [ X11 ]   [ U1 |    ] [  0  0  0 ]
X = [-----] = [---------] [----------] V1**T .
    [ X21 ]   [    | U2 ] [  0  0  0 ]
                            [  0  S  0 ]
                            [  0  0  I2]
```

X11 is P-by-Q. The orthogonal matrices U1, U2, and V1 are P-by-P, (M-P)-by-(M-P), and Q-by-Q, respectively. C and S are R-by-R nonnegative diagonal matrices satisfying C^2 + S^2 = I, in which R = MIN(P,M-P,Q,M-Q). I1 is a K1-by-K1 identity matrix and I2 is a K2-by-K2 identity matrix, where K1 = MAX(Q+P-M,0), K2 = MAX(Q-P,0).

Syntax

Fortran specification:

```
use armpl_library

subroutine dorcsd2by1(JOBU1, JOBU2, JOBV1T, M, P, Q, X11, LDX11, X21, LDX21,
                      THETA, U1, LDU1, U2, LDU2, V1T, LDV1T, WORK, LWORK,
                      IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dorcsd2by1_(const char *jobu1, const char *jobu2, const char *jobv1t,
                 const armpl_int_t *m, const armpl_int_t *p,
                 const armpl_int_t *q, double *x11, const armpl_int_t *ldx11,
                 double *x21, const armpl_int_t *ldx21, double *theta,
                 double *u1, const armpl_int_t *ldu1, double *u2,
                 const armpl_int_t *ldu2, double *v1t,
                 const armpl_int_t *ldv1t, double *work,
                 const armpl_int_t *lwork, armpl_int_t *iwork,
                 armpl_int_t *info, ... );
```

Parameters

JOBU1 Input parameter.

JOBU1 is CHARACTER

= 'Y': U1 is computed; otherwise: U1 is not computed.

JOBU2 Input parameter.

JOBU2 is CHARACTER

= 'Y': U2 is computed; otherwise: U2 is not computed.

JOBV1T Input parameter.

JOBV1T is CHARACTER

= 'Y': V1T is computed; otherwise: V1T is not computed.

M Input parameter.

M is INTEGER

The number of rows in X.

P Input parameter.

P is INTEGER

The number of rows in X11. 0 <= P <= M.

Q Input parameter.

Q is INTEGER

The number of columns in X11 and X21. 0 <= Q <= M.

X11 Input and output parameter.

X11 is DOUBLE PRECISION

X11 is an array, dimension (LDX11, Q). On entry, part of the orthogonal matrix whose CSD is desired.

LDX11 Input parameter.

LDX11 is INTEGER

The leading dimension of X11. LDX11 >= MAX(1, P).

X21 Input and output parameter.

X21 is DOUBLE PRECISION

X21 is an array, dimension (LDX21, Q). On entry, part of the orthogonal matrix whose CSD is desired.

LDX21 Input parameter.

LDX21 is INTEGER

The leading dimension of X21. LDX21 >= MAX(1,M-P).

THETA Output parameter.

THETA is DOUBLE PRECISION

THETA is an array, dimension (R), in which R =. MIN(P,M-P, Q,M-Q). C = DIAG(COS(THETA(1)), ... , COS(THETA(R))) and S = DIAG(SIN(THETA(1)), ... , SIN(THETA(R))).

U1 Output parameter.

U1 is DOUBLE PRECISION

U1 is an array, dimension (P). If JOBU1 = 'Y', U1 contains the P-by-P orthogonal matrix U1.

LDU1 Input parameter.

LDU1 is INTEGER

The leading dimension of U1. If JOBU1 = 'Y', LDU1 >= MAX(1, P).

U2 Output parameter.

U2 is DOUBLE PRECISION

U2 is an array, dimension (M-P). If JOBU2 = 'Y', U2 contains the (M-P)-by-(M-P) orthogonal matrix U2.

LDU2 Input parameter.

LDU2 is INTEGER

The leading dimension of U2. If JOBU2 = 'Y', LDU2 >= MAX(1,M-P).

V1T Output parameter.

V1T is DOUBLE PRECISION

V1T is an array, dimension (Q). If JOBV1T = 'Y', V1T contains the Q-by-Q matrix orthogonal matrix $V1^T$.

LDV1T Input parameter.

LDV1T is INTEGER

The leading dimension of V1T. If JOBV1T = 'Y', LDV1T >= MAX(1, Q).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK. If INFO > 0 on exit, WORK(2:R) contains the values PHI(1), . . . , PHI(R-1) that, together with THETA(1), . . . , THETA(R), define the matrix in intermediate bidiagonal-block form remaining after nonconvergence. INFO specifies the number of nonzero PHI's.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the work array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (M-MIN(P,M-P, Q,M-Q))

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: DBBCSD did not converge. See the description of WORK above for details.

Related Information

For this routine in other precisions, please see *sorcsd2by1*. It also exists with a native C interface as *LAPACKE_dorcsd2by1*.

4.8.9 sbbcsd

sbbcsd computes the CS decomposition of an orthogonal matrix in bidiagonal-block form,

```
[ B11 | B12 0  0 ]
[  0  |  0 -I  0 ]
```

X = [————————-]

```
[ B21 | B22 0  0 ]
[  0  |  0  0  I ]

                          [  C | -S  0  0 ]
            [ U1 |    ] [  0 |  0 -I  0 ] [ V1 |    ]**T
          = [---------] [---------------] [---------]   .
            [    | U2 ] [  S |  C  0  0 ] [    | V2 ]
                          [  0 |  0  0  I ]
```

X is M-by-M, its top-left block is P-by-Q, and Q must be no larger than P, M-P, or M-Q. (If Q is not the smallest index, then X must be transposed and/or permuted. This can be done in constant time using the TRANS and SIGNS options. See SORCSD for details.)

The bidiagonal matrices B11, B12, B21, and B22 are represented implicitly by angles THETA(1:Q) and PHI(1:Q-1).

The orthogonal matrices U1, U2, V1T, and V2T are input/output. The input matrices are pre- or post-multiplied by the appropriate singular vector matrices.

Syntax

Fortran specification:

```
use armpl_library

subroutine sbbcsd(JOBU1, JOBU2, JOBV1T, JOBV2T, TRANS, M, P, Q, THETA, PHI,
                  U1, LDU1, U2, LDU2, V1T, LDV1T, V2T, LDV2T, B11D, B11E,
                  B12D, B12E, B21D, B21E, B22D, B22E, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sbbcsd_(const char *jobu1, const char *jobu2, const char *jobv1t,
             const char *jobv2t, const char *trans, const armpl_int_t *m,
             const armpl_int_t *p, const armpl_int_t *q, float *theta,
             float *phi, float *u1, const armpl_int_t *ldu1, float *u2,
             const armpl_int_t *ldu2, float *v1t, const armpl_int_t *ldv1t,
             float *v2t, const armpl_int_t *ldv2t, float *b11d, float *b11e,
             float *b12d, float *b12e, float *b21d, float *b21e, float *b22d,
             float *b22e, float *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

JOBU1 Input parameter.

JOBU1 is CHARACTER

= 'Y': U1 is updated; otherwise: U1 is not updated.

JOBU2 Input parameter.

JOBU2 is CHARACTER

= 'Y': U2 is updated; otherwise: U2 is not updated.

JOBV1T Input parameter.

JOBV1T is CHARACTER

= 'Y': V1T is updated; otherwise: V1T is not updated.

JOBV2T Input parameter.

JOBV2T is CHARACTER

= 'Y': V2T is updated; otherwise: V2T is not updated.

TRANS Input parameter.

TRANS is CHARACTER

= 'T': X, U1, U2, V1T, and V2T are stored in row-major order; otherwise: X, U1, U2, V1T, and V2T are stored in column- major order.

M Input parameter.

M is INTEGER

The number of rows and columns in X, the orthogonal matrix in bidiagonal-block form.

P Input parameter.

P is INTEGER

The number of rows in the top-left block of X. 0 <= P <= M.

Q Input parameter.

Q is INTEGER

The number of columns in the top-left block of X. 0 <= Q <= MIN(P,M-P,M-Q).

THETA Input and output parameter.

THETA is REAL

THETA is an array, dimension (Q). On entry, the angles THETA(1),...,THETA(Q) that, along with PHI(1), ...,PHI(Q-1), define the matrix in bidiagonal-block form. On exit, the angles whose cosines and sines define the diagonal blocks in the CS decomposition.

PHI Input and output parameter.

PHI is REAL

PHI is an array, dimension (Q-1). The angles PHI(1),...,PHI(Q-1) that, along with THETA(1),..., THETA(Q), define the matrix in bidiagonal-block form.

U1 Input and output parameter.

U1 is REAL

U1 is an array, dimension (LDU1, P). On entry, a P-by-P matrix. On exit, U1 is postmultiplied by the left singular vector matrix common to [B11 ; 0] and [B12 0 0 ; 0 -I 0 0].

LDU1 Input parameter.

LDU1 is INTEGER

The leading dimension of the array U1, LDU1 >= MAX(1, P).

U2 Input and output parameter.

U2 is REAL

U2 is an array, dimension (LDU2,M-P). On entry, an (M-P)-by-(M-P) matrix. On exit, U2 is postmultiplied by the left singular vector matrix common to [B21 ; 0] and [B22 0 0 ; 0 0 I].

LDU2 Input parameter.

LDU2 is INTEGER

The leading dimension of the array U2, LDU2 >= MAX(1,M-P).

V1T Input and output parameter.

V1T is REAL

V1T is an array, dimension (LDV1T, Q). On entry, a Q-by-Q matrix. On exit, V1T is premultiplied by the transpose of the right singular vector matrix common to [B11 ; 0] and [B21 ; 0].

LDV1T Input parameter.

LDV1T is INTEGER

The leading dimension of the array V1T, LDV1T >= MAX(1, Q).

V2T Input and output parameter.

V2T is REAL

V2T is an array, dimension (LDV2T,M-Q). On entry, an (M-Q)-by-(M-Q) matrix. On exit, V2T is premultiplied by the transpose of the right singular vector matrix common to [B12 0 0 ; 0 -I 0] and [B22 0 0 ; 0 0 I].

LDV2T Input parameter.

LDV2T is INTEGER

The leading dimension of the array V2T, LDV2T >= MAX(1,M-Q).

B11D Output parameter.

B11D is REAL

B11D is an array, dimension (Q). When SBBCSD converges, B11D contains the cosines of THETA(1), ..., THETA(Q). If SBBCSD fails to converge, then B11D contains the diagonal of the partially reduced top-left block.

B11E Output parameter.

B11E is REAL

B11E is an array, dimension (Q-1). When SBBCSD converges, B11E contains zeros. If SBBCSD fails to converge, then B11E contains the superdiagonal of the partially reduced top-left block.

B12D Output parameter.

B12D is REAL

B12D is an array, dimension (Q). When SBBCSD converges, B12D contains the negative sines of THETA(1), ..., THETA(Q). If SBBCSD fails to converge, then B12D contains the diagonal of the partially reduced top-right block.

B12E Output parameter.

B12E is REAL

B12E is an array, dimension (Q-1). When SBBCSD converges, B12E contains zeros. If SBBCSD fails to converge, then B12E contains the subdiagonal of the partially reduced top-right block.

B21D Output parameter.

B21D is REAL

B21D is an array, dimension (Q). When SBBCSD converges, B21D contains the negative sines of THETA(1), ..., THETA(Q). If SBBCSD fails to converge, then B21D contains the diagonal of the partially reduced bottom-left block.

B21E Output parameter.

B21E is REAL

B21E is an array, dimension (Q-1). When SBBCSD converges, B21E contains zeros. If SBBCSD fails to converge, then B21E contains the subdiagonal of the partially reduced bottom-left block.

B22D Output parameter.

B22D is REAL

B22D is an array, dimension (Q). When SBBCSD converges, B22D contains the negative sines of THETA(1), ..., THETA(Q). If SBBCSD fails to converge, then B22D contains the diagonal of the partially reduced bottom-right block.

B22E Output parameter.

B22E is REAL

B22E is an array, dimension (Q-1). When SBBCSD converges, B22E contains zeros. If SBBCSD fails to converge, then B22E contains the subdiagonal of the partially reduced bottom-right block.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= MAX(1,8*Q).

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the work array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if SBBCSD did not converge, INFO specifies the number of nonzero entries in PHI, and B11D, B11E, etc., contain the partially reduced matrix.

Related Information

For this routine in other precisions, please see *cbbcsd*, *dbbcsd* and *zbbcsd*. It also exists with a native C interface as *LAPACKE_sbbcsd*.

4.8.10 sorbdb

sorbdb simultaneously bidiagonalizes the blocks of an M-by-M partitioned orthogonal matrix X:

```
                                [ B11 | B12 0  0 ]
[ X11 | X12 ]   [ P1 |    ] [  0  |  0 -I  0 ] [ Q1 |    ]**T
```

X = [————] = [———] [————————-] [———] .

```
[ X21 | X22 ]   [    | P2 ] [ B21 | B22 0  0 ] [    | Q2 ]
                           [  0  |  0  0  I ]
```

X11 is P-by-Q. Q must be no larger than P, M-P, or M-Q. (If this is not the case, then X must be transposed and/or permuted. This can be done in constant time using the TRANS and SIGNS options. See SORCSD for details.)

The orthogonal matrices P1, P2, Q1, and Q2 are P-by-P, (M-P)-by- (M-P), Q-by-Q, and (M-Q)-by-(M-Q), respectively. They are represented implicitly by Householder vectors.

B11, B12, B21, and B22 are Q-by-Q bidiagonal matrices represented implicitly by angles THETA, PHI.

Syntax

Fortran specification:

```
use armpl_library

subroutine sorbdb(TRANS, SIGNS, M, P, Q, X11, LDX11, X12, LDX12, X21, LDX21,
                  X22, LDX22, THETA, PHI, TAUP1, TAUP2, TAUQ1, TAUQ2, WORK,
                  LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sorbdb_(const char *trans, const char *signs, const armpl_int_t *m,
             const armpl_int_t *p, const armpl_int_t *q, float *x11,
             const armpl_int_t *ldx11, float *x12, const armpl_int_t *ldx12,
             float *x21, const armpl_int_t *ldx21, float *x22,
```

(continues on next page)

(continued from previous page)

```
            const armpl_int_t *ldx22, float *theta, float *phi, float *taup1,
            float *taup2, float *tauq1, float *tauq2, float *work,
            const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER

= 'T': X, U1, U2, V1T, and V2T are stored in row-major order; otherwise: X, U1, U2, V1T, and V2T are stored in column- major order.

SIGNS Input parameter.

SIGNS is CHARACTER

= 'O': The lower-left block is made nonpositive (the "other" convention); otherwise: The upper-right block is made nonpositive (the "default" convention).

M Input parameter.

M is INTEGER

The number of rows and columns in X.

P Input parameter.

P is INTEGER

The number of rows in X11 and X12. 0 <= P <= M.

Q Input parameter.

Q is INTEGER

The number of columns in X11 and X21. 0 <= Q <= MIN(P,M-P,M-Q).

X11 Input and output parameter.

X11 is REAL

X11 is an array, dimension (LDX11, Q). On entry, the top-left block of the orthogonal matrix to be reduced. On exit, the form depends on TRANS: If TRANS = 'N', then the columns of tril(X11) specify reflectors for P1, the rows of triu(X11,1) specify reflectors for Q1; else TRANS = 'T', and the rows of triu(X11) specify reflectors for P1, the columns of tril(X11,-1) specify reflectors for Q1.

LDX11 Input parameter.

LDX11 is INTEGER

The leading dimension of X11. If TRANS = 'N', then LDX11 >= P; else LDX11 >= Q.

X12 Input and output parameter.

X12 is REAL

X12 is an array, dimension (LDX12,M-Q). On entry, the top-right block of the orthogonal matrix to be reduced. On exit, the form depends on TRANS: If TRANS = 'N', then the rows of triu(X12) specify the first P reflectors for Q2; else TRANS = 'T', and the columns of tril(X12) specify the first P reflectors for Q2.

LDX12 Input parameter.

LDX12 is INTEGER

The leading dimension of X12. If TRANS = 'N', then LDX12 >= P; else LDX11 >= M-Q.

X21 Input and output parameter.

X21 is REAL

X21 is an array, dimension (LDX21, Q). On entry, the bottom-left block of the orthogonal matrix to be reduced. On exit, the form depends on TRANS: If TRANS = 'N', then the columns of tril(X21) specify reflectors for P2; else TRANS = 'T', and the rows of triu(X21) specify reflectors for P2.

LDX21 Input parameter.

LDX21 is INTEGER

The leading dimension of X21. If TRANS = 'N', then LDX21 >= M-P; else LDX21 >= Q.

X22 Input and output parameter.

X22 is REAL

X22 is an array, dimension (LDX22,M-Q). On entry, the bottom-right block of the orthogonal matrix to be reduced. On exit, the form depends on TRANS: If TRANS = 'N', then the rows of triu(X22(Q+1:M-P,P+1:M-Q)) specify the last M-P-Q reflectors for Q2, else TRANS = 'T', and the columns of tril(X22(P+1:M-Q,Q+1:M-P)) specify the last M-P-Q reflectors for P2.

LDX22 Input parameter.

LDX22 is INTEGER

The leading dimension of X22. If TRANS = 'N', then LDX22 >= M-P; else LDX22 >= M-Q.

THETA Output parameter.

THETA is REAL

THETA is an array, dimension (Q). The entries of the bidiagonal blocks B11, B12, B21, B22 can be computed from the angles THETA and PHI. See Further Details.

PHI Output parameter.

PHI is REAL

PHI is an array, dimension (Q-1). The entries of the bidiagonal blocks B11, B12, B21, B22 can be computed from the angles THETA and PHI. See Further Details.

TAUP1 Output parameter.

TAUP1 is REAL

TAUP1 is an array, dimension (P). The scalar factors of the elementary reflectors that define P1.

TAUP2 Output parameter.

TAUP2 is REAL

TAUP2 is an array, dimension (M-P). The scalar factors of the elementary reflectors that define P2.

TAUQ1 Output parameter.

TAUQ1 is REAL

TAUQ1 is an array, dimension (Q). The scalar factors of the elementary reflectors that define Q1.

TAUQ2 Output parameter.

TAUQ2 is REAL

TAUQ2 is an array, dimension (M-Q). The scalar factors of the elementary reflectors that define Q2.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= M-Q.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *dorbdb*. It also exists with a native C interface as *LAPACKE_sorbdb*.

4.8.11 sorcsd

sorcsd computes the CS decomposition of an M-by-M partitioned orthogonal matrix X:

```
                               [  I  0  0 |  0  0  0 ]
                               [  0  C  0 |  0 -S  0 ]
[ X11 | X12 ]   [ U1 |    ] [  0  0  0 |  0  0 -I ] [ V1 |    ]**T
```

X = [————] = [———] [————————] [———] .

```
[ X21 | X22 ]   [    | U2 ] [  0  0  0 |  I  0  0 ] [    | V2 ]
                           [  0  S  0 |  0  C  0 ]
                           [  0  0  I |  0  0  0 ]
```

X11 is P-by-Q. The orthogonal matrices U1, U2, V1, and V2 are P-by-P, (M-P)-by-(M-P), Q-by-Q, and (M-Q)-by-(M-Q), respectively. C and S are R-by-R nonnegative diagonal matrices satisfying C^2 + S^2 = I, in which R = MIN(P,M-P,Q,M-Q).

Syntax

Fortran specification:

```
use armpl_library

recursive subroutine sorcsd(JOBU1, JOBU2, JOBV1T, JOBV2T, TRANS, SIGNS, M, P,
                            Q, X11, LDX11, X12, LDX12, X21, LDX21, X22, LDX22,
                            THETA, U1, LDU1, U2, LDU2, V1T, LDV1T, V2T, LDV2T,
                            WORK, LWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sorcsd_(const char *jobu1, const char *jobu2, const char *jobv1t,
             const char *jobv2t, const char *trans, const char *signs,
             const armpl_int_t *m, const armpl_int_t *p, const armpl_int_t *q,
             float *x11, const armpl_int_t *ldx11, float *x12,
             const armpl_int_t *ldx12, float *x21, const armpl_int_t *ldx21,
             float *x22, const armpl_int_t *ldx22, float *theta, float *u1,
             const armpl_int_t *ldu1, float *u2, const armpl_int_t *ldu2,
             float *v1t, const armpl_int_t *ldv1t, float *v2t,
```

(continues on next page)

(continued from previous page)

```
            const armpl_int_t *ldv2t, float *work, const armpl_int_t *lwork,
            armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

JOBU1 Input parameter.

JOBU1 is CHARACTER

= 'Y': U1 is computed; otherwise: U1 is not computed.

JOBU2 Input parameter.

JOBU2 is CHARACTER

= 'Y': U2 is computed; otherwise: U2 is not computed.

JOBV1T Input parameter.

JOBV1T is CHARACTER

= 'Y': V1T is computed; otherwise: V1T is not computed.

JOBV2T Input parameter.

JOBV2T is CHARACTER

= 'Y': V2T is computed; otherwise: V2T is not computed.

TRANS Input parameter.

TRANS is CHARACTER

= 'T': X, U1, U2, V1T, and V2T are stored in row-major order; otherwise: X, U1, U2, V1T, and V2T are stored in column- major order.

SIGNS Input parameter.

SIGNS is CHARACTER

= 'O': The lower-left block is made nonpositive (the "other" convention); otherwise: The upper-right block is made nonpositive (the "default" convention).

M Input parameter.

M is INTEGER

The number of rows and columns in X.

P Input parameter.

P is INTEGER

The number of rows in X11 and X12. 0 <= P <= M.

Q Input parameter.

Q is INTEGER

The number of columns in X11 and X21. 0 <= Q <= M.

X11 Input and output parameter.

X11 is REAL

X11 is an array, dimension (LDX11, Q). On entry, part of the orthogonal matrix whose CSD is desired.

LDX11 Input parameter.

LDX11 is INTEGER

The leading dimension of X11. LDX11 >= MAX(1, P).

X12 Input and output parameter.

X12 is REAL

X12 is an array, dimension (LDX12,M-Q). On entry, part of the orthogonal matrix whose CSD is desired.

LDX12 Input parameter.

LDX12 is INTEGER

The leading dimension of X12. LDX12 >= MAX(1, P).

X21 Input and output parameter.

X21 is REAL

X21 is an array, dimension (LDX21, Q). On entry, part of the orthogonal matrix whose CSD is desired.

LDX21 Input parameter.

LDX21 is INTEGER

The leading dimension of X11. LDX21 >= MAX(1,M-P).

X22 Input and output parameter.

X22 is REAL

X22 is an array, dimension (LDX22,M-Q). On entry, part of the orthogonal matrix whose CSD is desired.

LDX22 Input parameter.

LDX22 is INTEGER

The leading dimension of X11. LDX22 >= MAX(1,M-P).

THETA Output parameter.

THETA is REAL

THETA is an array, dimension (R), in which R =. MIN(P,M-P, Q,M-Q). C = DIAG(COS(THETA(1)), ... , COS(THETA(R))) and S = DIAG(SIN(THETA(1)), ... , SIN(THETA(R))).

U1 Output parameter.

U1 is REAL

U1 is an array, dimension (LDU1, P). If JOBU1 = 'Y', U1 contains the P-by-P orthogonal matrix U1.

LDU1 Input parameter.

LDU1 is INTEGER

The leading dimension of U1. If JOBU1 = 'Y', LDU1 >= MAX(1, P).

U2 Output parameter.

U2 is REAL

U2 is an array, dimension (LDU2,M-P). If JOBU2 = 'Y', U2 contains the (M-P)-by-(M-P) orthogonal matrix U2.

LDU2 Input parameter.

LDU2 is INTEGER

The leading dimension of U2. If JOBU2 = 'Y', LDU2 >= MAX(1,M-P).

V1T Output parameter.

V1T is REAL

V1T is an array, dimension (LDV1T, Q). If JOBV1T = 'Y', V1T contains the Q-by-Q matrix orthogonal matrix $V1^T$.

LDV1T Input parameter.

LDV1T is INTEGER

The leading dimension of V1T. If JOBV1T = 'Y', LDV1T >= MAX(1, Q).

V2T Output parameter.

V2T is REAL

V2T is an array, dimension (LDV2T,M-Q). If JOBV2T = 'Y', V2T contains the (M-Q)-by-(M-Q) orthogonal matrix $V2^T$.

LDV2T Input parameter.

LDV2T is INTEGER

The leading dimension of V2T. If JOBV2T = 'Y', LDV2T >= MAX(1,M-Q).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK. If INFO > 0 on exit, WORK(2:R) contains the values PHI(1), ..., PHI(R-1) that, together with THETA(1), ..., THETA(R), define the matrix in intermediate bidiagonal-block form remaining after nonconvergence. INFO specifies the number of nonzero PHI's.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the work array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (M-MIN(P, M-P, Q, M-Q))

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: SBBCSD did not converge. See the description of WORK above for details.

Related Information

For this routine in other precisions, please see *dorcsd*. It also exists with a native C interface as *LAPACKE_sorcsd*.

4.8.12 sorcsd2by1

SORCSD2BY1 computes the CS decomposition of an M-by-Q matrix X with orthonormal columns that has been partitioned into a 2-by-1 block structure:

```
                            [  I1 0  0 ]
                            [  0  C  0 ]
    [ X11 ]   [ U1 |    ] [  0  0  0 ]
X = [-----] = [---------] [----------] V1**T .
    [ X21 ]   [    | U2 ] [  0  0  0 ]
                            [  0  S  0 ]
                            [  0  0  I2]
```

X11 is P-by-Q. The orthogonal matrices U1, U2, and V1 are P-by-P, (M-P)-by-(M-P), and Q-by-Q, respectively. C and S are R-by-R nonnegative diagonal matrices satisfying C^2 + S^2 = I, in which R = MIN(P,M-P,Q,M-Q). I1 is a K1-by-K1 identity matrix and I2 is a K2-by-K2 identity matrix, where K1 = MAX(Q+P-M,0), K2 = MAX(Q-P,0).

Syntax

Fortran specification:

```
use armpl_library

subroutine sorcsd2by1(JOBU1, JOBU2, JOBV1T, M, P, Q, X11, LDX11, X21, LDX21,
                      THETA, U1, LDU1, U2, LDU2, V1T, LDV1T, WORK, LWORK,
                      IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sorcsd2by1_(const char *jobu1, const char *jobu2, const char *jobv1t,
                 const armpl_int_t *m, const armpl_int_t *p,
                 const armpl_int_t *q, float *x11, const armpl_int_t *ldx11,
                 float *x21, const armpl_int_t *ldx21, float *theta,
                 float *u1, const armpl_int_t *ldu1, float *u2,
                 const armpl_int_t *ldu2, float *v1t,
                 const armpl_int_t *ldv1t, float *work,
                 const armpl_int_t *lwork, armpl_int_t *iwork,
                 armpl_int_t *info, ... );
```

Parameters

JOBU1 Input parameter.

JOBU1 is CHARACTER

= 'Y': U1 is computed; otherwise: U1 is not computed.

JOBU2 Input parameter.

JOBU2 is CHARACTER

= 'Y': U2 is computed; otherwise: U2 is not computed.

JOBV1T Input parameter.

JOBV1T is CHARACTER

= 'Y': V1T is computed; otherwise: V1T is not computed.

M Input parameter.

M is INTEGER

The number of rows in X.

P Input parameter.

P is INTEGER

The number of rows in X11. 0 <= P <= M.

Q Input parameter.

Q is INTEGER

The number of columns in X11 and X21. 0 <= Q <= M.

X11 Input and output parameter.

X11 is REAL

X11 is an array, dimension (LDX11, Q). On entry, part of the orthogonal matrix whose CSD is desired.

LDX11 Input parameter.

LDX11 is INTEGER

The leading dimension of X11. LDX11 >= MAX(1, P).

X21 Input and output parameter.

X21 is REAL

X21 is an array, dimension (LDX21, Q). On entry, part of the orthogonal matrix whose CSD is desired.

LDX21 Input parameter.

LDX21 is INTEGER

The leading dimension of X21. LDX21 >= MAX(1,M-P).

THETA Output parameter.

THETA is REAL

THETA is an array, dimension (R), in which R =. MIN(P,M-P, Q,M-Q). C = DIAG(COS(THETA(1)), ... , COS(THETA(R))) and S = DIAG(SIN(THETA(1)), ... , SIN(THETA(R))).

U1 Output parameter.

U1 is REAL

U1 is an array, dimension (P). If JOBU1 = 'Y', U1 contains the P-by-P orthogonal matrix U1.

LDU1 Input parameter.

LDU1 is INTEGER

The leading dimension of U1. If JOBU1 = 'Y', LDU1 >= MAX(1, P).

U2 Output parameter.

U2 is REAL

U2 is an array, dimension (M-P). If JOBU2 = 'Y', U2 contains the (M-P)-by-(M-P) orthogonal matrix U2.

LDU2 Input parameter.

LDU2 is INTEGER

The leading dimension of U2. If JOBU2 = 'Y', LDU2 >= MAX(1,M-P).

V1T Output parameter.

V1T is REAL

V1T is an array, dimension (Q). If JOBV1T = 'Y', V1T contains the Q-by-Q matrix orthogonal matrix $V1^T$.

LDV1T Input parameter.

LDV1T is INTEGER

The leading dimension of V1T. If JOBV1T = 'Y', LDV1T >= MAX(1, Q).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK. If INFO > 0 on exit, WORK(2:R) contains the values PHI(1), ..., PHI(R-1) that, together with THETA(1), ..., THETA(R), define the matrix in intermediate bidiagonal-block form remaining after nonconvergence. INFO specifies the number of nonzero PHI's.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the work array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (M-MIN(P,M-P, Q,M-Q))

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: SBBCSD did not converge. See the description of WORK above for details.

Related Information

For this routine in other precisions, please see *dorcsd2by1*. It also exists with a native C interface as *LAPACKE_sorcsd2by1*.

4.8.13 zbbcsd

zbbcsd computes the CS decomposition of a unitary matrix in bidiagonal-block form,

```
[ B11 | B12 0  0 ]
[  0  |  0 -I  0 ]
```

X = [————————-]

```
[ B21 | B22 0  0 ]
[  0  |  0  0  I ]

                           [  C | -S  0  0 ]
             [ U1 |    ] [  0 |  0 -I  0 ] [ V1 |    ]**H
           = [---------] [---------------] [---------]   .
             [    | U2 ] [  S |  C  0  0 ] [    | V2 ]
                           [  0 |  0  0  I ]
```

X is M-by-M, its top-left block is P-by-Q, and Q must be no larger than P, M-P, or M-Q. (If Q is not the smallest index, then X must be transposed and/or permuted. This can be done in constant time using the TRANS and SIGNS options. See ZUNCSD for details.)

The bidiagonal matrices B11, B12, B21, and B22 are represented implicitly by angles THETA(1:Q) and PHI(1:Q-1).

The unitary matrices U1, U2, V1T, and V2T are input/output. The input matrices are pre- or post-multiplied by the appropriate singular vector matrices.

Syntax

Fortran specification:

```
use armpl_library

subroutine zbbcsd(JOBU1, JOBU2, JOBV1T, JOBV2T, TRANS, M, P, Q, THETA, PHI,
                  U1, LDU1, U2, LDU2, V1T, LDV1T, V2T, LDV2T, B11D, B11E,
                  B12D, B12E, B21D, B21E, B22D, B22E, RWORK, LRWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zbbcsd_(const char *jobu1, const char *jobu2, const char *jobv1t,
             const char *jobv2t, const char *trans, const armpl_int_t *m,
             const armpl_int_t *p, const armpl_int_t *q, double *theta,
             double *phi, armpl_doublecomplex_t *u1, const armpl_int_t *ldu1,
             armpl_doublecomplex_t *u2, const armpl_int_t *ldu2,
             armpl_doublecomplex_t *v1t, const armpl_int_t *ldv1t,
             armpl_doublecomplex_t *v2t, const armpl_int_t *ldv2t,
             double *b11d, double *b11e, double *b12d, double *b12e,
             double *b21d, double *b21e, double *b22d, double *b22e,
             double *rwork, const armpl_int_t *lrwork, armpl_int_t *info,
             ... );
```

Parameters

JOBU1 Input parameter.

JOBU1 is CHARACTER

= 'Y': U1 is updated; otherwise: U1 is not updated.

JOBU2 Input parameter.

JOBU2 is CHARACTER

= 'Y': U2 is updated; otherwise: U2 is not updated.

JOBV1T Input parameter.

JOBV1T is CHARACTER

= 'Y': V1T is updated; otherwise: V1T is not updated.

JOBV2T Input parameter.

JOBV2T is CHARACTER

= 'Y': V2T is updated; otherwise: V2T is not updated.

TRANS Input parameter.

TRANS is CHARACTER

= 'T': X, U1, U2, V1T, and V2T are stored in row-major order; otherwise: X, U1, U2, V1T, and V2T are stored in column- major order.

M Input parameter.

M is INTEGER

The number of rows and columns in X, the unitary matrix in bidiagonal-block form.

P Input parameter.

P is INTEGER

The number of rows in the top-left block of X. 0 <= P <= M.

Q Input parameter.

Q is INTEGER

The number of columns in the top-left block of X. 0 <= Q <= MIN(P,M-P,M-Q).

THETA Input and output parameter.

THETA is DOUBLE PRECISION

THETA is an array, dimension (Q). On entry, the angles THETA(1),...,THETA(Q) that, along with PHI(1), ...,PHI(Q-1), define the matrix in bidiagonal-block form. On exit, the angles whose cosines and sines define the diagonal blocks in the CS decomposition.

PHI Input and output parameter.

PHI is DOUBLE PRECISION

PHI is an array, dimension (Q-1). The angles PHI(1),...,PHI(Q-1) that, along with THETA(1),..., THETA(Q), define the matrix in bidiagonal-block form.

U1 Input and output parameter.

U1 is COMPLEX*16

U1 is an array, dimension (LDU1, P). On entry, a P-by-P matrix. On exit, U1 is postmultiplied by the left singular vector matrix common to [B11 ; 0] and [B12 0 0 ; 0 -I 0 0].

LDU1 Input parameter.

LDU1 is INTEGER

The leading dimension of the array U1, LDU1 >= MAX(1, P).

U2 Input and output parameter.

U2 is COMPLEX*16

U2 is an array, dimension (LDU2,M-P). On entry, an (M-P)-by-(M-P) matrix. On exit, U2 is postmultiplied by the left singular vector matrix common to [B21 ; 0] and [B22 0 0 ; 0 0 I].

LDU2 Input parameter.

LDU2 is INTEGER

The leading dimension of the array U2, LDU2 >= MAX(1,M-P).

V1T Input and output parameter.

V1T is COMPLEX*16

V1T is an array, dimension (LDV1T, Q). On entry, a Q-by-Q matrix. On exit, V1T is premultiplied by the conjugate transpose of the right singular vector matrix common to [B11 ; 0] and [B21 ; 0].

LDV1T Input parameter.

LDV1T is INTEGER

The leading dimension of the array V1T, LDV1T >= MAX(1, Q).

V2T Input and output parameter.

V2T is COMPLEX*16

V2T is an array, dimension (LDV2T,M-Q). On entry, an (M-Q)-by-(M-Q) matrix. On exit, V2T is premultiplied by the conjugate transpose of the right singular vector matrix common to [B12 0 0 ; 0 -I 0] and [B22 0 0 ; 0 0 I].

LDV2T Input parameter.

LDV2T is INTEGER

The leading dimension of the array V2T, LDV2T >= MAX(1,M-Q).

B11D Output parameter.

B11D is DOUBLE PRECISION

B11D is an array, dimension (Q). When ZBBCSD converges, B11D contains the cosines of THETA(1), ..., THETA(Q). If ZBBCSD fails to converge, then B11D contains the diagonal of the partially reduced top-left block.

B11E Output parameter.

B11E is DOUBLE PRECISION

B11E is an array, dimension (Q-1). When ZBBCSD converges, B11E contains zeros. If ZBBCSD fails to converge, then B11E contains the superdiagonal of the partially reduced top-left block.

B12D Output parameter.

B12D is DOUBLE PRECISION

B12D is an array, dimension (Q). When ZBBCSD converges, B12D contains the negative sines of THETA(1), ..., THETA(Q). If ZBBCSD fails to converge, then B12D contains the diagonal of the partially reduced top-right block.

B12E Output parameter.

B12E is DOUBLE PRECISION

B12E is an array, dimension (Q-1). When ZBBCSD converges, B12E contains zeros. If ZBBCSD fails to converge, then B12E contains the subdiagonal of the partially reduced top-right block.

B21D Output parameter.

B21D is DOUBLE PRECISION

B21D is an array, dimension (Q). When ZBBCSD converges, B21D contains the negative sines of THETA(1), ..., THETA(Q). If ZBBCSD fails to converge, then B21D contains the diagonal of the partially reduced bottom-left block.

B21E Output parameter.

B21E is DOUBLE PRECISION

B21E is an array, dimension (Q-1). When ZBBCSD converges, B21E contains zeros. If ZBBCSD fails to converge, then B21E contains the subdiagonal of the partially reduced bottom-left block.

B22D Output parameter.

B22D is DOUBLE PRECISION

B22D is an array, dimension (Q). When ZBBCSD converges, B22D contains the negative sines of THETA(1), ..., THETA(Q). If ZBBCSD fails to converge, then B22D contains the diagonal of the partially reduced bottom-right block.

B22E Output parameter.

B22E is DOUBLE PRECISION

B22E is an array, dimension (Q-1). When ZBBCSD converges, B22E contains zeros. If ZBBCSD fails to converge, then B22E contains the subdiagonal of the partially reduced bottom-right block.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (MAX(1, LRWORK)). On exit, if INFO = 0, RWORK(1) returns the optimal LRWORK.

LRWORK Input parameter.

LRWORK is INTEGER

The dimension of the array RWORK. LRWORK >= MAX(1,8*Q).

If LRWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the RWORK array, returns this value as the first entry of the work array, and no error message related to LRWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if ZBBCSD did not converge, INFO specifies the number of nonzero entries in PHI, and B11D, B11E, etc., contain the partially reduced matrix.

Related Information

For this routine in other precisions, please see *cbbcsd*, *dbbcsd* and *sbbcsd*. It also exists with a native C interface as *LAPACKE_zbbcsd*.

4.8.14 zunbdb

zunbdb simultaneously bidiagonalizes the blocks of an M-by-M partitioned unitary matrix X:

```
                                [ B11 | B12 0  0 ]
[ X11 | X12 ]   [ P1 |    ] [  0  |  0 -I  0 ] [ Q1 |    ]**H
```

X = [————] = [————] [————————-] [————] .

```
[ X21 | X22 ]   [    | P2 ] [ B21 | B22 0  0 ] [    | Q2 ]
                                [  0  |  0  0  I ]
```

X11 is P-by-Q. Q must be no larger than P, M-P, or M-Q. (If this is not the case, then X must be transposed and/or permuted. This can be done in constant time using the TRANS and SIGNS options. See ZUNCSD for details.)

The unitary matrices P1, P2, Q1, and Q2 are P-by-P, (M-P)-by- (M-P), Q-by-Q, and (M-Q)-by-(M-Q), respectively. They are represented implicitly by Householder vectors.

B11, B12, B21, and B22 are Q-by-Q bidiagonal matrices represented implicitly by angles THETA, PHI.

Syntax

Fortran specification:

```
use armpl_library

subroutine zunbdb(TRANS, SIGNS, M, P, Q, X11, LDX11, X12, LDX12, X21, LDX21,
                  X22, LDX22, THETA, PHI, TAUP1, TAUP2, TAUQ1, TAUQ2, WORK,
                  LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zunbdb_(const char *trans, const char *signs, const armpl_int_t *m,
             const armpl_int_t *p, const armpl_int_t *q,
             armpl_doublecomplex_t *x11, const armpl_int_t *ldx11,
             armpl_doublecomplex_t *x12, const armpl_int_t *ldx12,
             armpl_doublecomplex_t *x21, const armpl_int_t *ldx21,
             armpl_doublecomplex_t *x22, const armpl_int_t *ldx22,
             double *theta, double *phi, armpl_doublecomplex_t *taup1,
             armpl_doublecomplex_t *taup2, armpl_doublecomplex_t *tauq1,
             armpl_doublecomplex_t *tauq2, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER

= 'T': X, U1, U2, V1T, and V2T are stored in row-major order; otherwise: X, U1, U2, V1T, and V2T are stored in column- major order.

SIGNS Input parameter.

SIGNS is CHARACTER

= 'O': The lower-left block is made nonpositive (the "other" convention); otherwise: The upper-right block is made nonpositive (the "default" convention).

M Input parameter.

M is INTEGER

The number of rows and columns in X.

P Input parameter.

P is INTEGER

The number of rows in X11 and X12. 0 <= P <= M.

Q Input parameter.

Q is INTEGER

The number of columns in X11 and X21. 0 <= Q <= MIN(P,M-P,M-Q).

X11 Input and output parameter.

X11 is COMPLEX*16

X11 is an array, dimension (LDX11, Q). On entry, the top-left block of the unitary matrix to be reduced. On exit, the form depends on TRANS: If TRANS = 'N', then the columns of tril(X11) specify reflectors for P1, the rows of triu(X11,1) specify reflectors for Q1; else TRANS = 'T', and the rows of triu(X11) specify reflectors for P1, the columns of tril(X11,-1) specify reflectors for Q1.

LDX11 Input parameter.

LDX11 is INTEGER

The leading dimension of X11. If TRANS = 'N', then LDX11 >= P; else LDX11 >= Q.

X12 Input and output parameter.

X12 is COMPLEX*16

X12 is an array, dimension (LDX12,M-Q). On entry, the top-right block of the unitary matrix to be reduced. On exit, the form depends on TRANS: If TRANS = 'N', then the rows of triu(X12) specify the first P reflectors for Q2; else TRANS = 'T', and the columns of tril(X12) specify the first P reflectors for Q2.

LDX12 Input parameter.

LDX12 is INTEGER

The leading dimension of X12. If TRANS = 'N', then LDX12 >= P; else LDX11 >= M-Q.

X21 Input and output parameter.

X21 is COMPLEX*16

X21 is an array, dimension (LDX21, Q). On entry, the bottom-left block of the unitary matrix to be reduced. On exit, the form depends on TRANS: If TRANS = 'N', then the columns of tril(X21) specify reflectors for P2; else TRANS = 'T', and the rows of triu(X21) specify reflectors for P2.

LDX21 Input parameter.

LDX21 is INTEGER

The leading dimension of X21. If TRANS = 'N', then LDX21 >= M-P; else LDX21 >= Q.

X22 Input and output parameter.

X22 is COMPLEX*16

X22 is an array, dimension (LDX22,M-Q). On entry, the bottom-right block of the unitary matrix to be reduced. On exit, the form depends on TRANS: If TRANS = 'N', then the rows of triu(X22(Q+1:M-P,P+1:M-Q)) specify the last M-P-Q reflectors for Q2, else TRANS = 'T', and the columns of tril(X22(P+1:M-Q,Q+1:M-P)) specify the last M-P-Q reflectors for P2.

LDX22 Input parameter.

LDX22 is INTEGER

The leading dimension of X22. If TRANS = 'N', then LDX22 >= M-P; else LDX22 >= M-Q.

THETA Output parameter.

THETA is DOUBLE PRECISION

THETA is an array, dimension (Q). The entries of the bidiagonal blocks B11, B12, B21, B22 can be computed from the angles THETA and PHI. See Further Details.

PHI Output parameter.

PHI is DOUBLE PRECISION

PHI is an array, dimension (Q-1). The entries of the bidiagonal blocks B11, B12, B21, B22 can be computed from the angles THETA and PHI. See Further Details.

TAUP1 Output parameter.

TAUP1 is COMPLEX*16

TAUP1 is an array, dimension (P). The scalar factors of the elementary reflectors that define P1.

TAUP2 Output parameter.

TAUP2 is COMPLEX*16

TAUP2 is an array, dimension (M-P). The scalar factors of the elementary reflectors that define P2.

TAUQ1 Output parameter.

TAUQ1 is COMPLEX*16

TAUQ1 is an array, dimension (Q). The scalar factors of the elementary reflectors that define Q1.

TAUQ2 Output parameter.

TAUQ2 is COMPLEX*16

TAUQ2 is an array, dimension (M-Q). The scalar factors of the elementary reflectors that define Q2.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= M-Q.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cunbdb*. It also exists with a native C interface as *LAPACKE_zunbdb*.

4.8.15 zuncsd

`zuncsd` computes the CS decomposition of an M-by-M partitioned unitary matrix X:

```
                                [  I  0  0 |  0  0  0 ]
                                [  0  C  0 |  0 -S  0 ]
[ X11 | X12 ]   [ U1 |    ] [  0  0  0 |  0  0 -I ] [ V1 |    ]**H
```

X = [————] = [———] [—————————] [———] .

```
[ X21 | X22 ]   [    | U2 ] [  0  0  0 |  I  0  0 ] [    | V2 ]
                                [  0  S  0 |  0  C  0 ]
                                [  0  0  I |  0  0  0 ]
```

X11 is P-by-Q. The unitary matrices U1, U2, V1, and V2 are P-by-P, (M-P)-by-(M-P), Q-by-Q, and (M-Q)-by-(M-Q), respectively. C and S are R-by-R nonnegative diagonal matrices satisfying C^2 + S^2 = I, in which R = MIN(P,M-P,Q,M-Q).

Syntax

Fortran specification:

```
use armpl_library

recursive subroutine zuncsd(JOBU1, JOBU2, JOBV1T, JOBV2T, TRANS, SIGNS, M, P,
                            Q, X11, LDX11, X12, LDX12, X21, LDX21, X22, LDX22,
                            THETA, U1, LDU1, U2, LDU2, V1T, LDV1T, V2T, LDV2T,
                            WORK, LWORK, RWORK, LRWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zuncsd_(const char *jobu1, const char *jobu2, const char *jobv1t,
             const char *jobv2t, const char *trans, const char *signs,
             const armpl_int_t *m, const armpl_int_t *p, const armpl_int_t *q,
             armpl_doublecomplex_t *x11, const armpl_int_t *ldx11,
             armpl_doublecomplex_t *x12, const armpl_int_t *ldx12,
             armpl_doublecomplex_t *x21, const armpl_int_t *ldx21,
             armpl_doublecomplex_t *x22, const armpl_int_t *ldx22,
             double *theta, armpl_doublecomplex_t *u1,
             const armpl_int_t *ldu1, armpl_doublecomplex_t *u2,
             const armpl_int_t *ldu2, armpl_doublecomplex_t *v1t,
             const armpl_int_t *ldv1t, armpl_doublecomplex_t *v2t,
             const armpl_int_t *ldv2t, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, double *rwork,
             const armpl_int_t *lrwork, armpl_int_t *iwork, armpl_int_t *info,
             ... );
```

Parameters

JOBU1 Input parameter.

`JOBU1` is CHARACTER

= 'Y': `U1` is computed; otherwise: `U1` is not computed.

JOBU2 Input parameter.

JOBU2 is CHARACTER

= 'Y': U2 is computed; otherwise: U2 is not computed.

JOBV1T Input parameter.

JOBV1T is CHARACTER

= 'Y': V1T is computed; otherwise: V1T is not computed.

JOBV2T Input parameter.

JOBV2T is CHARACTER

= 'Y': V2T is computed; otherwise: V2T is not computed.

TRANS Input parameter.

TRANS is CHARACTER

= 'T': X, U1, U2, V1T, and V2T are stored in row-major order; otherwise: X, U1, U2, V1T, and V2T are stored in column- major order.

SIGNS Input parameter.

SIGNS is CHARACTER

= 'O': The lower-left block is made nonpositive (the "other" convention); otherwise: The upper-right block is made nonpositive (the "default" convention).

M Input parameter.

M is INTEGER

The number of rows and columns in X.

P Input parameter.

P is INTEGER

The number of rows in X11 and X12. 0 <= P <= M.

Q Input parameter.

Q is INTEGER

The number of columns in X11 and X21. 0 <= Q <= M.

X11 Input and output parameter.

X11 is COMPLEX*16

X11 is an array, dimension (LDX11, Q). On entry, part of the unitary matrix whose CSD is desired.

LDX11 Input parameter.

LDX11 is INTEGER

The leading dimension of X11. LDX11 >= MAX(1, P).

X12 Input and output parameter.

X12 is COMPLEX*16

X12 is an array, dimension (LDX12,M-Q). On entry, part of the unitary matrix whose CSD is desired.

LDX12 Input parameter.

LDX12 is INTEGER

The leading dimension of X12. LDX12 >= MAX(1, P).

X21 Input and output parameter.

X21 is COMPLEX*16

X21 is an array, dimension (LDX21, Q). On entry, part of the unitary matrix whose CSD is desired.

LDX21 Input parameter.

LDX21 is INTEGER

The leading dimension of X11. LDX21 >= MAX(1,M-P).

X22 Input and output parameter.

X22 is COMPLEX*16

X22 is an array, dimension (LDX22,M-Q). On entry, part of the unitary matrix whose CSD is desired.

LDX22 Input parameter.

LDX22 is INTEGER

The leading dimension of X11. LDX22 >= MAX(1,M-P).

THETA Output parameter.

THETA is DOUBLE PRECISION

THETA is an array, dimension (R), in which R =. MIN(P,M-P, Q,M-Q). C = DIAG(COS(THETA(1)), ... , COS(THETA(R))) and S = DIAG(SIN(THETA(1)), ... , SIN(THETA(R))).

U1 Output parameter.

U1 is COMPLEX*16

U1 is an array, dimension (LDU1, P). If JOBU1 = 'Y', U1 contains the P-by-P unitary matrix U1.

LDU1 Input parameter.

LDU1 is INTEGER

The leading dimension of U1. If JOBU1 = 'Y', LDU1 >= MAX(1, P).

U2 Output parameter.

U2 is COMPLEX*16

U2 is an array, dimension (LDU2,M-P). If JOBU2 = 'Y', U2 contains the (M-P)-by-(M-P) unitary matrix U2.

LDU2 Input parameter.

LDU2 is INTEGER

The leading dimension of U2. If JOBU2 = 'Y', LDU2 >= MAX(1,M-P).

V1T Output parameter.

V1T is COMPLEX*16

V1T is an array, dimension (LDV1T, Q). If JOBV1T = 'Y', V1T contains the Q-by-Q matrix unitary matrix $V1^H$.

LDV1T Input parameter.

LDV1T is INTEGER

The leading dimension of V1T. If JOBV1T = 'Y', LDV1T >= MAX(1, Q).

V2T Output parameter.

V2T is COMPLEX*16

V2T is an array, dimension (LDV2T,M-Q). If JOBV2T = 'Y', V2T contains the (M-Q)-by-(M-Q) unitary matrix $V2^H$.

LDV2T Input parameter.

LDV2T is INTEGER

The leading dimension of V2T. If JOBV2T = 'Y', LDV2T >= MAX(1,M-Q).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the work array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension MAX(1, LRWORK). On exit, if INFO = 0, RWORK(1) returns the optimal LRWORK. If INFO > 0 on exit, RWORK(2:R) contains the values PHI(1), ..., PHI(R-1) that, together with THETA(1), ..., THETA(R), define the matrix in intermediate bidiagonal-block form remaining after nonconvergence. INFO specifies the number of nonzero PHI's.

LRWORK Input parameter.

LRWORK is INTEGER

The dimension of the array RWORK.

If LRWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the RWORK array, returns this value as the first entry of the work array, and no error message related to LRWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (M-MIN(P,M-P, Q,M-Q))

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: ZBBCSD did not converge. See the description of RWORK above for details.

Related Information

For this routine in other precisions, please see *cuncsd*. It also exists with a native C interface as *LAPACKE_zuncsd*.

4.8.16 zuncsd2by1

ZUNCSD2BY1 computes the CS decomposition of an M-by-Q matrix X with orthonormal columns that has been partitioned into a 2-by-1 block structure:

```
                           [  I1 0  0 ]
                           [  0  C  0 ]
    [ X11 ]   [ U1 |    ] [  0  0  0 ]
X = [-----] = [---------] [----------] V1**T .
    [ X21 ]   [    | U2 ] [  0  0  0 ]
```

(continues on next page)

(continued from previous page)

```
                          [  0  S  0 ]
                          [  0  0  I2]
```

X11 is P-by-Q. The unitary matrices U1, U2, and V1 are P-by-P, (M-P)-by-(M-P), and Q-by-Q, respectively. C and S are R-by-R nonnegative diagonal matrices satisfying C^2 + S^2 = I, in which R = MIN(P,M-P,Q,M-Q). I1 is a K1-by-K1 identity matrix and I2 is a K2-by-K2 identity matrix, where K1 = MAX(Q+P-M,0), K2 = MAX(Q-P,0).

Syntax

Fortran specification:

```
use armpl_library

subroutine zuncsd2by1(JOBU1, JOBU2, JOBV1T, M, P, Q, X11, LDX11, X21, LDX21,
                      THETA, U1, LDU1, U2, LDU2, V1T, LDV1T, WORK, LWORK,
                      RWORK, LRWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zuncsd2by1_(const char *jobu1, const char *jobu2, const char *jobv1t,
                 const armpl_int_t *m, const armpl_int_t *p,
                 const armpl_int_t *q, armpl_doublecomplex_t *x11,
                 const armpl_int_t *ldx11, armpl_doublecomplex_t *x21,
                 const armpl_int_t *ldx21, double *theta,
                 armpl_doublecomplex_t *u1, const armpl_int_t *ldu1,
                 armpl_doublecomplex_t *u2, const armpl_int_t *ldu2,
                 armpl_doublecomplex_t *v1t, const armpl_int_t *ldv1t,
                 armpl_doublecomplex_t *work, const armpl_int_t *lwork,
                 double *rwork, const armpl_int_t *lrwork, armpl_int_t *iwork,
                 armpl_int_t *info, ... );
```

Parameters

JOBU1 Input parameter.

JOBU1 is CHARACTER

= 'Y': U1 is computed; otherwise: U1 is not computed.

JOBU2 Input parameter.

JOBU2 is CHARACTER

= 'Y': U2 is computed; otherwise: U2 is not computed.

JOBV1T Input parameter.

JOBV1T is CHARACTER

= 'Y': V1T is computed; otherwise: V1T is not computed.

M Input parameter.

M is INTEGER

The number of rows in X.

P Input parameter.

P is INTEGER

The number of rows in X11. 0 <= P <= M.

Q Input parameter.

Q is INTEGER

The number of columns in X11 and X21. 0 <= Q <= M.

X11 Input and output parameter.

X11 is COMPLEX*16

X11 is an array, dimension (LDX11, Q). On entry, part of the unitary matrix whose CSD is desired.

LDX11 Input parameter.

LDX11 is INTEGER

The leading dimension of X11. LDX11 >= MAX(1, P).

X21 Input and output parameter.

X21 is COMPLEX*16

X21 is an array, dimension (LDX21, Q). On entry, part of the unitary matrix whose CSD is desired.

LDX21 Input parameter.

LDX21 is INTEGER

The leading dimension of X21. LDX21 >= MAX(1,M-P).

THETA Output parameter.

THETA is DOUBLE PRECISION

THETA is an array, dimension (R), in which R =. MIN(P,M-P, Q,M-Q). C = DIAG(COS(THETA(1)), ... , COS(THETA(R))) and S = DIAG(SIN(THETA(1)), ... , SIN(THETA(R))).

U1 Output parameter.

U1 is COMPLEX*16

U1 is an array, dimension (P). If JOBU1 = 'Y', U1 contains the P-by-P unitary matrix U1.

LDU1 Input parameter.

LDU1 is INTEGER

The leading dimension of U1. If JOBU1 = 'Y', LDU1 >= MAX(1, P).

U2 Output parameter.

U2 is COMPLEX*16

U2 is an array, dimension (M-P). If JOBU2 = 'Y', U2 contains the (M-P)-by-(M-P) unitary matrix U2.

LDU2 Input parameter.

LDU2 is INTEGER

The leading dimension of U2. If JOBU2 = 'Y', LDU2 >= MAX(1,M-P).

V1T Output parameter.

V1T is COMPLEX*16

V1T is an array, dimension (Q). If JOBV1T = 'Y', V1T contains the Q-by-Q matrix unitary matrix $V1^T$.

LDV1T Input parameter.

LDV1T is INTEGER

The leading dimension of V1T. If JOBV1T = 'Y', LDV1T >= MAX(1, Q).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the work array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (MAX(1, LRWORK)). On exit, if INFO = 0, RWORK(1) returns the optimal LRWORK. If INFO > 0 on exit, RWORK(2:R) contains the values PHI(1), ..., PHI(R-1) that, together with THETA(1), ..., THETA(R), define the matrix in intermediate bidiagonal-block form remaining after nonconvergence. INFO specifies the number of nonzero PHI's.

LRWORK Input parameter.

LRWORK is INTEGER

The dimension of the array RWORK.

If LRWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the RWORK array, returns this value as the first entry of the work array, and no error message related to LRWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (M-MIN(P,M-P, Q,M-Q))

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: ZBBCSD did not converge. See the description of WORK above for details.

Related Information

For this routine in other precisions, please see *cuncsd2by1*. It also exists with a native C interface as *LAPACKE_zuncsd2by1*.

4.9 LAPACK solution refinement routines

4.9.1 cgbrfs

cgbrfs improves the computed solution to a system of linear equations when the coefficient matrix is banded, and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgbrfs(TRANS, N, KL, KU, NRHS, AB, LDAB, AFB, LDAFB, IPIV, B, LDB,
                  X, LDX, FERR, BERR, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgbrfs_(const char *trans, const armpl_int_t *n, const armpl_int_t *kl,
             const armpl_int_t *ku, const armpl_int_t *nrhs,
             const armpl_singlecomplex_t *ab, const armpl_int_t *ldab,
             const armpl_singlecomplex_t *afb, const armpl_int_t *ldafb,
             const armpl_int_t *ipiv, const armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, armpl_singlecomplex_t *x,
             const armpl_int_t *ldx, float *ferr, float *berr,
             armpl_singlecomplex_t *work, float *rwork, armpl_int_t *info,
             ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose)

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AB Input parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). The original band matrix A, stored in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(ku+1+i-j,j) = A(i,j) for max(1,j-ku)<=i<=min(n,j+kl).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KL+KU+1.

AFB Input parameter.

AFB is COMPLEX

AFB is an array, dimension (LDAFB, N). Details of the LU factorization of the band matrix A, as computed by CGBTRF. U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1.

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AFB. LDAFB >= 2*KL*KU+1.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from CGBTRF; for 1<=i<=N, row i of the matrix was interchanged with row IPIV(i).

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is COMPLEX

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by CGBTRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dgbrfs*, *sgbrfs* and *zgbrfs*. It also exists with a native C interface as *LAPACKE_cgbrfs*.

4.9.2 cgbrfsx

```
CGBRFSX improves the computed solution to a system of linear
   equations and provides error bounds and backward error estimates
   for the solution.  In addition to normwise error bound, the code
   provides maximum componentwise error bound if possible.  See
   comments for ERR_BNDS_NORM and ERR_BNDS_COMP for details of the
   error bounds.

   The original system of linear equations may have been equilibrated
   before calling this routine, as described by arguments EQUED, R
   and C below. In this case, the solution and error bounds returned
   are for the original unequilibrated system.                    Some␣
↪optional parameters are bundled in the PARAMS array.  These
    settings determine how refinement is performed, but often the
    defaults are acceptable.  If the defaults are acceptable, users
    can pass NPARAMS = 0 which prevents the source code from accessing
    the PARAMS argument.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine cgbrfsx(TRANS, EQUED, N, KL, KU, NRHS, AB, LDAB, AFB, LDAFB, IPIV,
                   R, C, B, LDB, X, LDX, RCOND, BERR, N_ERR_BNDS,
                   ERR_BNDS_NORM, ERR_BNDS_COMP, NPARAMS, PARAMS, WORK, RWORK,
                   INFO)
```

C specification:

```
#include "armpl.h"

void cgbrfsx_(const char *trans, const char *equed, const armpl_int_t *n,
              const armpl_int_t *kl, const armpl_int_t *ku,
              const armpl_int_t *nrhs, const armpl_singlecomplex_t *ab,
              const armpl_int_t *ldab, const armpl_singlecomplex_t *afb,
              const armpl_int_t *ldafb, const armpl_int_t *ipiv,
              const float *r, const float *c, const armpl_singlecomplex_t *b,
              const armpl_int_t *ldb, armpl_singlecomplex_t *x,
              const armpl_int_t *ldx, float *rcond, float *berr,
              const armpl_int_t *n_err_bnds, float *err_bnds_norm,
              float *err_bnds_comp, const armpl_int_t *nparams, float *params,
              armpl_singlecomplex_t *work, float *rwork, armpl_int_t *info,
              ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose = Transpose)

EQUED Input parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done to A before calling this routine. This is needed to compute the solution and error bounds correctly. = 'N': No equilibration = 'R': Row equilibration, i.e., A has been premultiplied by diag(R). = 'C': Column equilibration, i.e., A has been postmultiplied by diag(C). = 'B': Both row and column equilibration, i.e., A has been replaced by diag(R) * A * diag(C). The right hand side B has been changed accordingly.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AB Input parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). The original band matrix A, stored in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(ku+1+i-j,j) = A(i,j) for max(1,j-ku)<=i<=min(n,j+kl).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KL+KU+1.

AFB Input parameter.

AFB is COMPLEX

AFB is an array, dimension (LDAFB, N). Details of the LU factorization of the band matrix A, as computed by DGBTRF. U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1.

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AFB. LDAFB >= 2*KL*KU+1.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from SGETRF; for 1<=i<=N, row i of the matrix was interchanged with row IPIV(i).

R Input and output parameter.

R is REAL

R is an array, dimension (N). The row scale factors for A. If EQUED = 'R' or 'B', A is multiplied on the left by diag(R); if EQUED = 'N' or 'C', R is not accessed. R is an input argument if FACT = 'F'; otherwise, R is an output argument. If FACT = 'F' and EQUED = 'R' or 'B', each element of R must be positive. If R is output, each element of R is a power of the radix. If R is input, each element of R should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

C Input and output parameter.

C is REAL

C is an array, dimension (N). The column scale factors for A. If EQUED = 'C' or 'B', A is multiplied on the right by diag(C); if EQUED = 'N' or 'R', C is not accessed. C is an input argument if FACT = 'F'; otherwise, C is an output argument. If FACT = 'F' and EQUED = 'C' or 'B', each element of C must be positive. If C is output, each element of C is a power of the radix. If C is input, each element of C should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is COMPLEX

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by SGETRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). Componentwise relative backward error. This is the componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

N_ERR_BNDS Input parameter.

N_ERR_BNDS is INTEGER

Number of error bounds to return for each right hand side and each type (normwise or componentwise). See ERR_BNDS_NORM and ERR_BNDS_COMP below.

ERR_BNDS_NORM Output parameter.

ERR_BNDS_NORM is REAL

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Output parameter.

ERR_BNDS_COMP is REAL

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:, N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix

Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

NPARAMS Input parameter.

NPARAMS is INTEGER

Specifies the number of parameters set in PARAMS. If .LE. 0, the PARAMS array is never referenced and default values are used.

PARAMS Input and output parameter.

PARAMS is REAL

PARAMS is an array, dimension NPARAMS. Specifies algorithm parameters. If an entry is .LT. 0.0, then that entry will be filled with default value used for that parameter. Only positions up to NPARAMS are accessed; defaults are used for higher-numbered parameters.

PARAMS(LA_LINRX_ITREF_I = 1) : Whether to perform iterative refinement or not. Default: 1.0 = 0.0 : No refinement is performed, and no error bounds are computed. = 1.0 : Use the double-precision refinement algorithm, possibly with doubled-single computations if the compilation environment does not support DOUBLE PRECISION. (other values are reserved for future use)

PARAMS(LA_LINRX_ITHRESH_I = 2) : Maximum number of residual computations allowed for refinement. Default: 10 Aggressive: Set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in err_bnds_norm and err_bnds_comp may no longer be trustworthy.

PARAMS(LA_LINRX_CWISE_I = 3) : Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Default: 1.0 (attempt componentwise convergence)

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. The solution to every right-hand side is guaranteed. < 0: If INFO = -i, the i-th argument had an illegal value > 0 and <= N: U(INFO, INFO) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+J: The solution corresponding to the Jth right-hand side is not guaranteed. The solutions corresponding to other right- hand sides K with K > J may not be guaranteed as well, but only the first such right-hand side is reported. If a small componentwise error is not requested (PARAMS(3) = 0.0) then the Jth right-hand side is the first with a normwise error bound that is not guaranteed (the smallest J such that ERR_BNDS_NORM(J,1) = 0.0). By default (PARAMS(3) = 1.0) the Jth right-hand side is the first with either a normwise or componentwise error bound that is not guaranteed (the smallest J such that either ERR_BNDS_NORM(J,1) = 0.0 or ERR_BNDS_COMP(J,1) = 0.0). See the definition of ERR_BNDS_NORM(:,1) and ERR_BNDS_COMP(:,1). To get information about all of the right-hand sides check ERR_BNDS_NORM or ERR_BNDS_COMP.

Related Information

For this routine in other precisions, please see *dgbrfsx*, *sgbrfsx* and *zgbrfsx*. It also exists with a native C interface as *LAPACKE_cgbrfsx*.

4.9.3 cgerfs

`cgerfs` improves the computed solution to a system of linear equations and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgerfs(TRANS, N, NRHS, A, LDA, AF, LDAF, IPIV, B, LDB, X, LDX,
                  FERR, BERR, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgerfs_(const char *trans, const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const armpl_singlecomplex_t *af, const armpl_int_t *ldaf,
             const armpl_int_t *ipiv, const armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, armpl_singlecomplex_t *x,
             const armpl_int_t *ldx, float *ferr, float *berr,
             armpl_singlecomplex_t *work, float *rwork, armpl_int_t *info,
             ... );
```

Parameters

TRANS Input parameter.

`TRANS` is `CHARACTER*1`

Specifies the form of the system of equations: = 'N': `A` * `X` = `B` (No transpose) = 'T': A^T * `X` = `B` (Transpose) = 'C': A^H * `X` = `B` (Conjugate transpose)

N Input parameter.

`N` is `INTEGER`

The order of the matrix `A`. `N` >= 0.

NRHS Input parameter.

`NRHS` is `INTEGER`

The number of right hand sides, i.e., the number of columns of the matrices `B` and `X`. `NRHS` >= 0.

A Input parameter.

`A` is `COMPLEX`

`A` is an array, dimension (`LDA`, `N`). The original N-by-N matrix `A`.

LDA Input parameter.

`LDA` is `INTEGER`

The leading dimension of the array `A`. `LDA` >= max(1, `N`).

AF Input parameter.

`AF` is `COMPLEX`

`AF` is an array, dimension (`LDAF`, `N`). The factors L and U from the factorization `A` = P*L*U as computed by CGETRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from CGETRF; for 1<=i<=N, row i of the matrix was interchanged with row IPIV(i).

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is COMPLEX

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by CGETRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dgerfs*, *sgerfs* and *zgerfs*. It also exists with a native C interface as *LAPACKE_cgerfs*.

4.9.4 cgerfsx

```
CGERFSX improves the computed solution to a system of linear
   equations and provides error bounds and backward error estimates
   for the solution.  In addition to normwise error bound, the code
   provides maximum componentwise error bound if possible.  See
   comments for ERR_BNDS_NORM and ERR_BNDS_COMP for details of the
   error bounds.

   The original system of linear equations may have been equilibrated
   before calling this routine, as described by arguments EQUED, R
   and C below. In this case, the solution and error bounds returned
   are for the original unequilibrated system.                    Some␣
↪optional parameters are bundled in the PARAMS array.  These
    settings determine how refinement is performed, but often the
    defaults are acceptable.  If the defaults are acceptable, users
    can pass NPARAMS = 0 which prevents the source code from accessing
    the PARAMS argument.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine cgerfsx(TRANS, EQUED, N, NRHS, A, LDA, AF, LDAF, IPIV, R, C, B,
                   LDB, X, LDX, RCOND, BERR, N_ERR_BNDS, ERR_BNDS_NORM,
                   ERR_BNDS_COMP, NPARAMS, PARAMS, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgerfsx_(const char *trans, const char *equed, const armpl_int_t *n,
              const armpl_int_t *nrhs, const armpl_singlecomplex_t *a,
              const armpl_int_t *lda, const armpl_singlecomplex_t *af,
              const armpl_int_t *ldaf, const armpl_int_t *ipiv,
              const float *r, const float *c, const armpl_singlecomplex_t *b,
              const armpl_int_t *ldb, armpl_singlecomplex_t *x,
              const armpl_int_t *ldx, float *rcond, float *berr,
              const armpl_int_t *n_err_bnds, float *err_bnds_norm,
              float *err_bnds_comp, const armpl_int_t *nparams, float *params,
              armpl_singlecomplex_t *work, float *rwork, armpl_int_t *info,
              ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose = Transpose)

EQUED Input parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done to A before calling this routine. This is needed to compute the solution and error bounds correctly. = 'N': No equilibration = 'R': Row equilibration, i.e., A has been premultiplied by diag(R). = 'C': Column equilibration, i.e., A has been postmultiplied by diag(C). = 'B': Both row and column equilibration, i.e., A has been replaced by diag(R) * A * diag(C). The right hand side B has been changed accordingly.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The original N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX

AF is an array, dimension (LDAF, N). The factors L and U from the factorization A = P*L*U as computed by CGETRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from CGETRF; for 1<=i<=N, row i of the matrix was interchanged with row IPIV(i).

R Input parameter.

R is REAL

R is an array, dimension (N). The row scale factors for A. If EQUED = 'R' or 'B', A is multiplied on the left by diag(R); if EQUED = 'N' or 'C', R is not accessed. If R is accessed, each element of R should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

C Input parameter.

C is REAL

C is an array, dimension (N). The column scale factors for A. If EQUED = 'C' or 'B', A is multiplied on the right by diag(C); if EQUED = 'N' or 'R', C is not accessed. If C is accessed, each element of C should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is COMPLEX

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by CGETRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). Componentwise relative backward error. This is the componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

N_ERR_BNDS Input parameter.

N_ERR_BNDS is INTEGER

Number of error bounds to return for each right hand side and each type (normwise or componentwise). See ERR_BNDS_NORM and ERR_BNDS_COMP below.

ERR_BNDS_NORM Output parameter.

ERR_BNDS_NORM is REAL

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 “Trust/don’t trust” boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch(‘Epsilon’).

err = 2 “Guaranteed” error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch(‘Epsilon’). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Output parameter.

ERR_BNDS_COMP is REAL

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:, N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

NPARAMS Input parameter.

NPARAMS is INTEGER

Specifies the number of parameters set in PARAMS. If .LE. 0, the PARAMS array is never referenced and default values are used.

PARAMS Input and output parameter.

PARAMS is REAL

PARAMS is an array, dimension NPARAMS. Specifies algorithm parameters. If an entry is .LT. 0.0, then that entry will be filled with default value used for that parameter. Only positions up to NPARAMS are accessed; defaults are used for higher-numbered parameters.

PARAMS(LA_LINRX_ITREF_I = 1) : Whether to perform iterative refinement or not. Default: 1.0 = 0.0 : No refinement is performed, and no error bounds are computed. = 1.0 : Use the double-precision refinement algorithm, possibly with doubled-single computations if the compilation environment does not support DOUBLE PRECISION. (other values are reserved for future use)

PARAMS(LA_LINRX_ITHRESH_I = 2) : Maximum number of residual computations allowed for refinement. Default: 10 Aggressive: Set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in err_bnds_norm and err_bnds_comp may no longer be trustworthy.

PARAMS(LA_LINRX_CWISE_I = 3) : Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Default: 1.0 (attempt componentwise convergence)

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. The solution to every right-hand side is guaranteed. < 0: If INFO = -i, the i-th argument had an illegal value > 0 and <= N: U(INFO, INFO) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+J: The solution corresponding to the Jth right-hand side is not guaranteed. The solutions corresponding to other right- hand sides K with K > J may not be guaranteed as well, but only the first such right-hand side is reported. If a small componentwise error is not requested (PARAMS(3) = 0.0) then the Jth right-hand side is the first with a normwise error bound that is not guaranteed (the smallest J such that ERR_BNDS_NORM(J,1) = 0.0). By default (PARAMS(3) = 1.0) the Jth right-hand side is the first with either a normwise or componentwise error bound that is not guaranteed (the smallest J such that either ERR_BNDS_NORM(J,1) = 0.0 or ERR_BNDS_COMP(J,1) = 0.0). See the definition of ERR_BNDS_NORM(:,1) and ERR_BNDS_COMP(:,1). To get information about all of the right-hand sides check ERR_BNDS_NORM or ERR_BNDS_COMP.

Related Information

For this routine in other precisions, please see *dgerfsx*, *sgerfsx* and *zgerfsx*. It also exists with a native C interface as *LAPACKE_cgerfsx*.

4.9.5 cgtrfs

cgtrfs improves the computed solution to a system of linear equations when the coefficient matrix is tridiagonal, and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgtrfs(TRANS, N, NRHS, DL, D, DU, DLF, DF, DUF, DU2, IPIV, B, LDB,
                  X, LDX, FERR, BERR, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgtrfs_(const char *trans, const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_singlecomplex_t *dl, const armpl_singlecomplex_t *d,
             const armpl_singlecomplex_t *du,
             const armpl_singlecomplex_t *dlf,
             const armpl_singlecomplex_t *df,
```

(continues on next page)

(continued from previous page)

```
                const armpl_singlecomplex_t *duf,
                const armpl_singlecomplex_t *du2, const armpl_int_t *ipiv,
                const armpl_singlecomplex_t *b, const armpl_int_t *ldb,
                armpl_singlecomplex_t *x, const armpl_int_t *ldx, float *ferr,
                float *berr, armpl_singlecomplex_t *work, float *rwork,
                armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose)

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

DL Input parameter.

DL is COMPLEX

DL is an array, dimension (N-1). The (n-1) subdiagonal elements of A.

D Input parameter.

D is COMPLEX

D is an array, dimension (N). The diagonal elements of A.

DU Input parameter.

DU is COMPLEX

DU is an array, dimension (N-1). The (n-1) superdiagonal elements of A.

DLF Input parameter.

DLF is COMPLEX

DLF is an array, dimension (N-1). The (n-1) multipliers that define the matrix L from the LU factorization of A as computed by CGTTRF.

DF Input parameter.

DF is COMPLEX

DF is an array, dimension (N). The n diagonal elements of the upper triangular matrix U from the LU factorization of A.

DUF Input parameter.

DUF is COMPLEX

DUF is an array, dimension (N-1). The (n-1) elements of the first superdiagonal of U.

DU2 Input parameter.

DU2 is COMPLEX

DU2 is an array, dimension (N-2). The (n-2) elements of the second superdiagonal of U.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices; for 1 <= i <= n, row i of the matrix was interchanged with row IPIV(i). IPIV(i) will always be either i or i+1; IPIV(i) = i indicates a row interchange was not required.

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is COMPLEX

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by CGTTRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dgtrfs*, *sgtrfs* and *zgtrfs*. It also exists with a native C interface as *LAPACKE_cgtrfs*.

4.9.6 cherfs

`cherfs` improves the computed solution to a system of linear equations when the coefficient matrix is Hermitian indefinite, and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine cherfs(UPLO, N, NRHS, A, LDA, AF, LDAF, IPIV, B, LDB, X, LDX, FERR,
                  BERR, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cherfs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const armpl_singlecomplex_t *af, const armpl_int_t *ldaf,
             const armpl_int_t *ipiv, const armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, armpl_singlecomplex_t *x,
             const armpl_int_t *ldx, float *ferr, float *berr,
             armpl_singlecomplex_t *work, float *rwork, armpl_int_t *info,
             ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX

AF is an array, dimension (LDAF, N). The factored form of the matrix A. AF contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^H or A = L*D*L^H as computed by CHETRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CHETRF.

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is COMPLEX

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by CHETRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zherfs*. It also exists with a native C interface as *LAPACKE_cherfs*.

4.9.7 cherfsx

```
CHERFSX improves the computed solution to a system of linear
   equations when the coefficient matrix is Hermitian indefinite, and
   provides error bounds and backward error estimates for the
   solution.  In addition to normwise error bound, the code provides
   maximum componentwise error bound if possible.  See comments for
   ERR_BNDS_NORM and ERR_BNDS_COMP for details of the error bounds.

   The original system of linear equations may have been equilibrated
   before calling this routine, as described by arguments EQUED and S
   below. In this case, the solution and error bounds returned are
   for the original unequilibrated system.                          Some optional␣
↪parameters are bundled in the PARAMS array.  These
    settings determine how refinement is performed, but often the
    defaults are acceptable.  If the defaults are acceptable, users
    can pass NPARAMS = 0 which prevents the source code from accessing
    the PARAMS argument.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine cherfsx(UPLO, EQUED, N, NRHS, A, LDA, AF, LDAF, IPIV, S, B, LDB, X,
                   LDX, RCOND, BERR, N_ERR_BNDS, ERR_BNDS_NORM, ERR_BNDS_COMP,
                   NPARAMS, PARAMS, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cherfsx_(const char *uplo, const char *equed, const armpl_int_t *n,
              const armpl_int_t *nrhs, const armpl_singlecomplex_t *a,
              const armpl_int_t *lda, const armpl_singlecomplex_t *af,
              const armpl_int_t *ldaf, const armpl_int_t *ipiv,
              const float *s, const armpl_singlecomplex_t *b,
              const armpl_int_t *ldb, armpl_singlecomplex_t *x,
              const armpl_int_t *ldx, float *rcond, float *berr,
              const armpl_int_t *n_err_bnds, float *err_bnds_norm,
              float *err_bnds_comp, const armpl_int_t *nparams, float *params,
              armpl_singlecomplex_t *work, float *rwork, armpl_int_t *info,
              ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

EQUED Input parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done to A before calling this routine. This is needed to compute the solution and error bounds correctly. = 'N': No equilibration = 'Y': Both row and column equilibration, i.e., A has been replaced by diag(S) * A * diag(S). The right hand side B has been changed accordingly.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX

AF is an array, dimension (LDAF, N). The factored form of the matrix A. AF contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by SSYTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by SSYTRF.

S Input and output parameter.

S is REAL

S is an array, dimension (N). The scale factors for A. If EQUED = 'Y', A is multiplied on the left and right by diag(S). S is an input argument if FACT = 'F'; otherwise, S is an output argument. If FACT = 'F' and EQUED = 'Y', each element of S must be positive. If S is output, each element of S is a power of the radix. If S is input, each element of S should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is COMPLEX

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by SGETRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). Componentwise relative backward error. This is the componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

N_ERR_BNDS Input parameter.

N_ERR_BNDS is INTEGER

Number of error bounds to return for each right hand side and each type (normwise or componentwise). See ERR_BNDS_NORM and ERR_BNDS_COMP below.

ERR_BNDS_NORM Output parameter.

ERR_BNDS_NORM is REAL

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 “Trust/don’t trust” boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch(‘Epsilon’).

err = 2 “Guaranteed” error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch(‘Epsilon’). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch(‘Epsilon’) to determine if the error estimate is “guaranteed”. These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Output parameter.

ERR_BNDS_COMP is REAL

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:, N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

NPARAMS Input parameter.

NPARAMS is INTEGER

Specifies the number of parameters set in PARAMS. If .LE. 0, the PARAMS array is never referenced and default values are used.

PARAMS Input and output parameter.

PARAMS is REAL

PARAMS is an array, dimension NPARAMS. Specifies algorithm parameters. If an entry is .LT. 0.0, then that entry will be filled with default value used for that parameter. Only positions up to NPARAMS are accessed; defaults are used for higher-numbered parameters.

PARAMS(LA_LINRX_ITREF_I = 1) : Whether to perform iterative refinement or not. Default: 1.0 = 0.0 : No refinement is performed, and no error bounds are computed. = 1.0 : Use the double-precision refinement algorithm, possibly with doubled-single computations if the compilation environment does not support DOUBLE PRECISION. (other values are reserved for future use)

PARAMS(LA_LINRX_ITHRESH_I = 2) : Maximum number of residual computations allowed for refinement. Default: 10 Aggressive: Set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in err_bnds_norm and err_bnds_comp may no longer be trustworthy.

PARAMS(LA_LINRX_CWISE_I = 3) : Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Default: 1.0 (attempt componentwise convergence)

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. The solution to every right-hand side is guaranteed. < 0: If INFO = -i, the i-th argument had an illegal value > 0 and <= N: U(INFO, INFO) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+J: The solution corresponding to the Jth right-hand side is not guaranteed. The solutions corresponding to other right- hand sides K with K > J may not be guaranteed as well, but only the first such right-hand side is reported. If a small componentwise error is not requested (PARAMS(3) = 0.0) then the Jth right-hand side is the first with a normwise error bound that is not guaranteed (the smallest J such that ERR_BNDS_NORM(J,1) = 0.0). By default (PARAMS(3) = 1.0) the Jth right-hand side is the first with either a normwise or componentwise error bound that is not guaranteed (the smallest J such that either ERR_BNDS_NORM(J,1) = 0.0 or ERR_BNDS_COMP(J,1) = 0.0). See the definition of ERR_BNDS_NORM(:,1) and ERR_BNDS_COMP(:,1). To get information about all of the right-hand sides check ERR_BNDS_NORM or ERR_BNDS_COMP.

Related Information

For this routine in other precisions, please see *zherfsx*. It also exists with a native C interface as *LAPACKE_cherfsx*.

4.9.8 chprfs

chprfs improves the computed solution to a system of linear equations when the coefficient matrix is Hermitian indefinite and packed, and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine chprfs(UPLO, N, NRHS, AP, AFP, IPIV, B, LDB, X, LDX, FERR, BERR,
                  WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void chprfs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_singlecomplex_t *ap,
             const armpl_singlecomplex_t *afp, const armpl_int_t *ipiv,
             const armpl_singlecomplex_t *b, const armpl_int_t *ldb,
             armpl_singlecomplex_t *x, const armpl_int_t *ldx, float *ferr,
             float *berr, armpl_singlecomplex_t *work, float *rwork,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AP Input parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). The upper or lower triangle of the Hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n.

AFP Input parameter.

AFP is COMPLEX

AFP is an array, dimension (N*(N+1)/2). The factored form of the matrix A. AFP contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^H or A = L*D*L^H as computed by CHPTRF, stored as a packed triangular matrix.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CHPTRF.

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is COMPLEX

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by CHPTRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zhprfs*. It also exists with a native C interface as *LAPACKE_chprfs*.

4.9.9 cpbrfs

cpbrfs improves the computed solution to a system of linear equations when the coefficient matrix is Hermitian positive definite and banded, and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine cpbrfs(UPLO, N, KD, NRHS, AB, LDAB, AFB, LDAFB, B, LDB, X, LDX,
                  FERR, BERR, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cpbrfs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
             const armpl_int_t *nrhs, const armpl_singlecomplex_t *ab,
             const armpl_int_t *ldab, const armpl_singlecomplex_t *afb,
             const armpl_int_t *ldafb, const armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, armpl_singlecomplex_t *x,
             const armpl_int_t *ldx, float *ferr, float *berr,
             armpl_singlecomplex_t *work, float *rwork, armpl_int_t *info,
             ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AB Input parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). The upper or lower triangle of the Hermitian band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

AFB Input parameter.

AFB is COMPLEX

AFB is an array, dimension (LDAFB, N). The triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H of the band matrix A as computed by CPBTRF, in the same storage format as A (see AB).

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AFB. LDAFB >= KD+1.

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is COMPLEX

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by CPBTRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dpbrfs*, *spbrfs* and *zpbrfs*. It also exists with a native C interface as *LAPACKE_cpbrfs*.

4.9.10 cporfs

cporfs improves the computed solution to a system of linear equations when the coefficient matrix is Hermitian positive definite, and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine cporfs(UPLO, N, NRHS, A, LDA, AF, LDAF, B, LDB, X, LDX, FERR, BERR,
                  WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cporfs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const armpl_singlecomplex_t *af, const armpl_int_t *ldaf,
             const armpl_singlecomplex_t *b, const armpl_int_t *ldb,
             armpl_singlecomplex_t *x, const armpl_int_t *ldx, float *ferr,
```

(continues on next page)

(continued from previous page)

```
            float *berr, armpl_singlecomplex_t *work, float *rwork,
            armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX

AF is an array, dimension (LDAF, N). The triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H , as computed by CPOTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is COMPLEX

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by CPOTRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dporfs*, *sporfs* and *zporfs*. It also exists with a native C interface as *LAPACKE_cporfs*.

4.9.11 cporfsx

```
CPORFSX improves the computed solution to a system of linear
   equations when the coefficient matrix is symmetric positive
   definite, and provides error bounds and backward error estimates
   for the solution.  In addition to normwise error bound, the code
   provides maximum componentwise error bound if possible.  See
   comments for ERR_BNDS_NORM and ERR_BNDS_COMP for details of the
   error bounds.

   The original system of linear equations may have been equilibrated
   before calling this routine, as described by arguments EQUED and S
   below. In this case, the solution and error bounds returned are
   for the original unequilibrated system.                          Some optional␣
↪parameters are bundled in the PARAMS array.  These
    settings determine how refinement is performed, but often the
    defaults are acceptable.  If the defaults are acceptable, users
    can pass NPARAMS = 0 which prevents the source code from accessing
    the PARAMS argument.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine cporfsx(UPLO, EQUED, N, NRHS, A, LDA, AF, LDAF, S, B, LDB, X, LDX,
                   RCOND, BERR, N_ERR_BNDS, ERR_BNDS_NORM, ERR_BNDS_COMP,
                   NPARAMS, PARAMS, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cporfsx_(const char *uplo, const char *equed, const armpl_int_t *n,
              const armpl_int_t *nrhs, const armpl_singlecomplex_t *a,
              const armpl_int_t *lda, const armpl_singlecomplex_t *af,
              const armpl_int_t *ldaf, const float *s,
              const armpl_singlecomplex_t *b, const armpl_int_t *ldb,
              armpl_singlecomplex_t *x, const armpl_int_t *ldx, float *rcond,
              float *berr, const armpl_int_t *n_err_bnds,
              float *err_bnds_norm, float *err_bnds_comp,
              const armpl_int_t *nparams, float *params,
              armpl_singlecomplex_t *work, float *rwork, armpl_int_t *info,
              ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

EQUED Input parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done to A before calling this routine. This is needed to compute the solution and error bounds correctly. = 'N': No equilibration = 'Y': Both row and column equilibration, i.e., A has been replaced by diag(S) * A * diag(S). The right hand side B has been changed accordingly.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX

AF is an array, dimension (LDAF, N). The triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , as computed by SPOTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

S Input and output parameter.

S is REAL

S is an array, dimension (N). The row scale factors for A. If EQUED = 'Y', A is multiplied on the left and right by diag(S). S is an input argument if FACT = 'F'; otherwise, S is an output argument. If FACT = 'F' and EQUED = 'Y', each element of S must be positive. If S is output, each element of S is a power of the radix. If S is input, each element of S should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is COMPLEX

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by SGETRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). Componentwise relative backward error. This is the componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

N_ERR_BNDS Input parameter.

N_ERR_BNDS is INTEGER

Number of error bounds to return for each right hand side and each type (normwise or componentwise). See ERR_BNDS_NORM and ERR_BNDS_COMP below.

ERR_BNDS_NORM Output parameter.

ERR_BNDS_NORM is REAL

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Output parameter.

ERR_BNDS_COMP is REAL

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:, N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

NPARAMS Input parameter.

NPARAMS is INTEGER

Specifies the number of parameters set in PARAMS. If .LE. 0, the PARAMS array is never referenced and default values are used.

PARAMS Input and output parameter.

PARAMS is REAL

PARAMS is an array, dimension NPARAMS. Specifies algorithm parameters. If an entry is .LT. 0.0, then that entry will be filled with default value used for that parameter. Only positions up to NPARAMS are accessed; defaults are used for higher-numbered parameters.

PARAMS(LA_LINRX_ITREF_I = 1) : Whether to perform iterative refinement or not. Default: 1.0 = 0.0 : No refinement is performed, and no error bounds are computed. = 1.0 : Use the double-precision refinement algorithm, possibly with doubled-single computations if the compilation environment does not support DOUBLE PRECISION. (other values are reserved for future use)

PARAMS(LA_LINRX_ITHRESH_I = 2) : Maximum number of residual computations allowed for refinement. Default: 10 Aggressive: Set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in err_bnds_norm and err_bnds_comp may no longer be trustworthy.

PARAMS(LA_LINRX_CWISE_I = 3) : Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Default: 1.0 (attempt componentwise convergence)

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. The solution to every right-hand side is guaranteed. < 0: If INFO = -i, the i-th argument had an illegal value > 0 and <= N: U(INFO, INFO) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+J: The solution corresponding to the Jth right-hand side is not guaranteed. The solutions corresponding to other right- hand sides K with K > J may not be guaranteed as well, but only the first such right-hand side is reported. If a small componentwise error is not requested (PARAMS(3) = 0.0) then the Jth right-hand side is the first with a normwise error bound that is not guaranteed (the smallest J such that ERR_BNDS_NORM(J,1) = 0.0). By default (PARAMS(3) = 1.0) the Jth right-hand side is the first with either a normwise or componentwise error bound that is not guaranteed (the smallest J such that either ERR_BNDS_NORM(J,1) = 0.0 or ERR_BNDS_COMP(J,1) = 0.0). See the definition of ERR_BNDS_NORM(:,1) and ERR_BNDS_COMP(:,1). To get information about all of the right-hand sides check ERR_BNDS_NORM or ERR_BNDS_COMP.

Related Information

For this routine in other precisions, please see *dporfsx*, *sporfsx* and *zporfsx*. It also exists with a native C interface as *LAPACKE_cporfsx*.

4.9.12 cpprfs

cpprfs improves the computed solution to a system of linear equations when the coefficient matrix is Hermitian positive definite and packed, and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine cpprfs(UPLO, N, NRHS, AP, AFP, B, LDB, X, LDX, FERR, BERR, WORK,
                  RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cpprfs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_singlecomplex_t *ap,
             const armpl_singlecomplex_t *afp, const armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, armpl_singlecomplex_t *x,
             const armpl_int_t *ldx, float *ferr, float *berr,
             armpl_singlecomplex_t *work, float *rwork, armpl_int_t *info,
             ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AP Input parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). The upper or lower triangle of the Hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

AFP Input parameter.

AFP is COMPLEX

AFP is an array, dimension (N*(N+1)/2). The triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H , as computed by SPPTRF/CPPTRF, packed columnwise in a linear array in the same format as A (see AP).

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is COMPLEX

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by CPPTRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dpprfs*, *spprfs* and *zpprfs*. It also exists with a native C interface as *LAPACKE_cpprfs*.

4.9.13 cptrfs

cptrfs improves the computed solution to a system of linear equations when the coefficient matrix is Hermitian positive definite and tridiagonal, and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine cptrfs(UPLO, N, NRHS, D, E, DF, EF, B, LDB, X, LDX, FERR, BERR,
                  WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cptrfs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const float *d, const armpl_singlecomplex_t *e, const float *df,
             const armpl_singlecomplex_t *ef, const armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, armpl_singlecomplex_t *x,
             const armpl_int_t *ldx, float *ferr, float *berr,
             armpl_singlecomplex_t *work, float *rwork, armpl_int_t *info,
             ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the superdiagonal or the subdiagonal of the tridiagonal matrix A is stored and the form of the factorization: = 'U': E is the superdiagonal of A, and A = U^H *D*U; = 'L': E is the subdiagonal of A, and A = L*D*L^H . (The two forms are equivalent if A is real.)

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

D Input parameter.

D is REAL

D is an array, dimension (N). The n real diagonal elements of the tridiagonal matrix A.

E Input parameter.

E is COMPLEX

E is an array, dimension (N-1). The (n-1) off-diagonal elements of the tridiagonal matrix A (see UPLO).

DF Input parameter.

DF is REAL

DF is an array, dimension (N). The n diagonal elements of the diagonal matrix D from the factorization computed by CPTTRF.

EF Input parameter.

EF is COMPLEX

EF is an array, dimension (N-1). The (n-1) off-diagonal elements of the unit bidiagonal factor U or L from the factorization computed by CPTTRF (see UPLO).

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is COMPLEX

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by CPTTRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j).

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dptrfs*, *sptrfs* and *zptrfs*. It also exists with a native C interface as *LAPACKE_cptrfs*.

4.9.14 csprfs

csprfs improves the computed solution to a system of linear equations when the coefficient matrix is symmetric indefinite and packed, and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine csprfs(UPLO, N, NRHS, AP, AFP, IPIV, B, LDB, X, LDX, FERR, BERR,
                  WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void csprfs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_singlecomplex_t *ap,
             const armpl_singlecomplex_t *afp, const armpl_int_t *ipiv,
             const armpl_singlecomplex_t *b, const armpl_int_t *ldb,
             armpl_singlecomplex_t *x, const armpl_int_t *ldx, float *ferr,
             float *berr, armpl_singlecomplex_t *work, float *rwork,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AP Input parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). The upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n.

AFP Input parameter.

AFP is COMPLEX

AFP is an array, dimension (N*(N+1)/2). The factored form of the matrix A. AFP contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by CSPTRF, stored as a packed triangular matrix.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CSPTRF.

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is COMPLEX

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by CSPTRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dsprfs*, *ssprfs* and *zsprfs*. It also exists with a native C interface as *LAPACKE_csprfs*.

4.9.15 csyrfs

csyrfs improves the computed solution to a system of linear equations when the coefficient matrix is symmetric indefinite, and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine csyrfs(UPLO, N, NRHS, A, LDA, AF, LDAF, IPIV, B, LDB, X, LDX, FERR,
                  BERR, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void csyrfs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const armpl_singlecomplex_t *af, const armpl_int_t *ldaf,
             const armpl_int_t *ipiv, const armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, armpl_singlecomplex_t *x,
             const armpl_int_t *ldx, float *ferr, float *berr,
             armpl_singlecomplex_t *work, float *rwork, armpl_int_t *info,
             ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX

AF is an array, dimension (LDAF, N). The factored form of the matrix A. AF contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by CSYTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CSYTRF.

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is COMPLEX

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by CSYTRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dsyrfs*, *ssyrfs* and *zsyrfs*. It also exists with a native C interface as *LAPACKE_csyrfs*.

4.9.16 csyrfsx

```
CSYRFSX improves the computed solution to a system of linear
   equations when the coefficient matrix is symmetric indefinite, and
   provides error bounds and backward error estimates for the
   solution.  In addition to normwise error bound, the code provides
   maximum componentwise error bound if possible.  See comments for
   ERR_BNDS_NORM and ERR_BNDS_COMP for details of the error bounds.

   The original system of linear equations may have been equilibrated
   before calling this routine, as described by arguments EQUED and S
   below. In this case, the solution and error bounds returned are
   for the original unequilibrated system.                    Some optional␣
↪parameters are bundled in the PARAMS array.  These
    settings determine how refinement is performed, but often the
    defaults are acceptable.  If the defaults are acceptable, users
    can pass NPARAMS = 0 which prevents the source code from accessing
    the PARAMS argument.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine csyrfsx(UPLO, EQUED, N, NRHS, A, LDA, AF, LDAF, IPIV, S, B, LDB, X,
                   LDX, RCOND, BERR, N_ERR_BNDS, ERR_BNDS_NORM, ERR_BNDS_COMP,
                   NPARAMS, PARAMS, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void csyrfsx_(const char *uplo, const char *equed, const armpl_int_t *n,
              const armpl_int_t *nrhs, const armpl_singlecomplex_t *a,
              const armpl_int_t *lda, const armpl_singlecomplex_t *af,
              const armpl_int_t *ldaf, const armpl_int_t *ipiv,
              const float *s, const armpl_singlecomplex_t *b,
              const armpl_int_t *ldb, armpl_singlecomplex_t *x,
              const armpl_int_t *ldx, float *rcond, float *berr,
              const armpl_int_t *n_err_bnds, float *err_bnds_norm,
              float *err_bnds_comp, const armpl_int_t *nparams, float *params,
              armpl_singlecomplex_t *work, float *rwork, armpl_int_t *info,
              ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

EQUED Input parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done to A before calling this routine. This is needed to compute the solution and error bounds correctly. = 'N': No equilibration = 'Y': Both row and column equilibration, i.e., A has been replaced by diag(S) * A * diag(S). The right hand side B has been changed accordingly.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX

AF is an array, dimension (LDAF, N). The factored form of the matrix A. AF contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by SSYTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by SSYTRF.

S Input and output parameter.

S is REAL

S is an array, dimension (N). The scale factors for A. If EQUED = 'Y', A is multiplied on the left and right by diag(S). S is an input argument if FACT = 'F'; otherwise, S is an output argument. If FACT = 'F' and EQUED = 'Y', each element of S must be positive. If S is output, each element of S is a power of the radix. If S is input, each element of S should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is COMPLEX

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by SGETRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). Componentwise relative backward error. This is the componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

N_ERR_BNDS Input parameter.

N_ERR_BNDS is INTEGER

Number of error bounds to return for each right hand side and each type (normwise or componentwise). See ERR_BNDS_NORM and ERR_BNDS_COMP below.

ERR_BNDS_NORM Output parameter.

ERR_BNDS_NORM is REAL

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 “Trust/don’t trust” boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch(‘Epsilon’).

err = 2 “Guaranteed” error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch(‘Epsilon’). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch(‘Epsilon’) to determine if the error estimate is “guaranteed”. These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Output parameter.

ERR_BNDS_COMP is REAL

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:, N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

NPARAMS Input parameter.

NPARAMS is INTEGER

Specifies the number of parameters set in PARAMS. If .LE. 0, the PARAMS array is never referenced and default values are used.

PARAMS Input and output parameter.

PARAMS is REAL

PARAMS is an array, dimension NPARAMS. Specifies algorithm parameters. If an entry is .LT. 0.0, then that entry will be filled with default value used for that parameter. Only positions up to NPARAMS are accessed; defaults are used for higher-numbered parameters.

PARAMS(LA_LINRX_ITREF_I = 1) : Whether to perform iterative refinement or not. Default: 1.0 = 0.0 : No refinement is performed, and no error bounds are computed. = 1.0 : Use the double-precision refinement algorithm, possibly with doubled-single computations if the compilation environment does not support DOUBLE PRECISION. (other values are reserved for future use)

PARAMS(LA_LINRX_ITHRESH_I = 2) : Maximum number of residual computations allowed for refinement. Default: 10 Aggressive: Set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in err_bnds_norm and err_bnds_comp may no longer be trustworthy.

PARAMS(LA_LINRX_CWISE_I = 3) : Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Default: 1.0 (attempt componentwise convergence)

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. The solution to every right-hand side is guaranteed. < 0: If INFO = -i, the i-th argument had an illegal value > 0 and <= N: U(INFO, INFO) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+J: The solution corresponding to the Jth right-hand side is not guaranteed. The solutions corresponding to other right- hand sides K with K > J may not be guaranteed as well, but only the first such right-hand side is reported. If a small componentwise error is not requested (PARAMS(3) = 0.0) then the Jth right-hand side is the first with a normwise error bound that is not guaranteed (the smallest J such that ERR_BNDS_NORM(J,1) = 0.0). By default (PARAMS(3) = 1.0) the Jth right-hand side is the first with either a normwise or componentwise error bound that is not guaranteed (the smallest J such that either ERR_BNDS_NORM(J,1) = 0.0 or ERR_BNDS_COMP(J,1) = 0.0). See the definition of ERR_BNDS_NORM(:,1) and ERR_BNDS_COMP(:,1). To get information about all of the right-hand sides check ERR_BNDS_NORM or ERR_BNDS_COMP.

Related Information

For this routine in other precisions, please see *dsyrfsx*, *ssyrfsx* and *zsyrfsx*. It also exists with a native C interface as *LAPACKE_csyrfsx*.

4.9.17 ctbrfs

ctbrfs provides error bounds and backward error estimates for the solution to a system of linear equations with a triangular band coefficient matrix.

The solution matrix X must be computed by CTBTRS or some other means before entering this routine. ctbrfs does not do iterative refinement because doing so cannot improve the backward error.

Syntax

Fortran specification:

```
use armpl_library

subroutine ctbrfs(UPLO, TRANS, DIAG, N, KD, NRHS, AB, LDAB, B, LDB, X, LDX,
                  FERR, BERR, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ctbrfs_(const char *uplo, const char *trans, const char *diag,
             const armpl_int_t *n, const armpl_int_t *kd,
             const armpl_int_t *nrhs, const armpl_singlecomplex_t *ab,
             const armpl_int_t *ldab, const armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, const armpl_singlecomplex_t *x,
             const armpl_int_t *ldx, float *ferr, float *berr,
             armpl_singlecomplex_t *work, float *rwork, armpl_int_t *info,
             ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose)

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals or subdiagonals of the triangular band matrix A. KD >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AB Input parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). The upper or lower triangular band matrix A, stored in the first kd+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd). If DIAG = 'U', the diagonal elements of A are not referenced and are assumed to be 1.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input parameter.

X is COMPLEX

X is an array, dimension (LDX, NRHS). The solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dtbrfs*, *stbrfs* and *ztbrfs*. It also exists with a native C interface as *LAPACKE_ctbrfs*.

4.9.18 ctprfs

ctprfs provides error bounds and backward error estimates for the solution to a system of linear equations with a triangular packed coefficient matrix.

The solution matrix X must be computed by CTPTRS or some other means before entering this routine. ctprfs does not do iterative refinement because doing so cannot improve the backward error.

Syntax

Fortran specification:

```
use armpl_library

subroutine ctprfs(UPLO, TRANS, DIAG, N, NRHS, AP, B, LDB, X, LDX, FERR, BERR,
                  WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ctprfs_(const char *uplo, const char *trans, const char *diag,
             const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_singlecomplex_t *ap, const armpl_singlecomplex_t *b,
```

(continues on next page)

(continued from previous page)

```
            const armpl_int_t *ldb, const armpl_singlecomplex_t *x,
            const armpl_int_t *ldx, float *ferr, float *berr,
            armpl_singlecomplex_t *work, float *rwork, armpl_int_t *info,
            ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose)

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AP Input parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). The upper or lower triangular matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n. If DIAG = 'U', the diagonal elements of A are not referenced and are assumed to be 1.

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input parameter.

X is COMPLEX

X is an array, dimension (LDX, NRHS). The solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dtprfs*, *stprfs* and *ztprfs*. It also exists with a native C interface as *LAPACKE_ctprfs*.

4.9.19 ctrrfs

ctrrfs provides error bounds and backward error estimates for the solution to a system of linear equations with a triangular coefficient matrix.

The solution matrix X must be computed by CTRTRS or some other means before entering this routine. ctrrfs does not do iterative refinement because doing so cannot improve the backward error.

Syntax

Fortran specification:

```
use armpl_library

subroutine ctrrfs(UPLO, TRANS, DIAG, N, NRHS, A, LDA, B, LDB, X, LDX, FERR,
                  BERR, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ctrrfs_(const char *uplo, const char *trans, const char *diag,
             const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda,
```

(continues on next page)

(continued from previous page)

```
                const armpl_singlecomplex_t *b, const armpl_int_t *ldb,
                const armpl_singlecomplex_t *x, const armpl_int_t *ldx,
                float *ferr, float *berr, armpl_singlecomplex_t *work,
                float *rwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose)

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input parameter.

X is COMPLEX

X is an array, dimension (LDX, NRHS). The solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dtrrfs*, *strrfs* and *ztrrfs*. It also exists with a native C interface as *LAPACKE_ctrrfs*.

4.9.20 dgbrfs

dgbrfs improves the computed solution to a system of linear equations when the coefficient matrix is banded, and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgbrfs(TRANS, N, KL, KU, NRHS, AB, LDAB, AFB, LDAFB, IPIV, B, LDB,
                  X, LDX, FERR, BERR, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgbrfs_(const char *trans, const armpl_int_t *n, const armpl_int_t *kl,
             const armpl_int_t *ku, const armpl_int_t *nrhs, const double *ab,
             const armpl_int_t *ldab, const double *afb,
             const armpl_int_t *ldafb, const armpl_int_t *ipiv,
             const double *b, const armpl_int_t *ldb, double *x,
             const armpl_int_t *ldx, double *ferr, double *berr, double *work,
             armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose = Transpose)

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AB Input parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). The original band matrix A, stored in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(ku+1+i-j,j) = A(i,j) for max(1,j-ku)<=i<=min(n,j+kl).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KL+KU+1.

AFB Input parameter.

AFB is DOUBLE PRECISION

AFB is an array, dimension (LDAFB, N). Details of the LU factorization of the band matrix A, as computed by DGBTRF. U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1.

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AFB. LDAFB >= 2*KL*KU+1.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from DGBTRF; for 1<=i<=N, row i of the matrix was interchanged with row IPIV(i).

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by DGBTRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgbrfs*, *sgbrfs* and *zgbrfs*. It also exists with a native C interface as *LAPACKE_dgbrfs*.

4.9.21 dgbrfsx

```
DGBRFSX improves the computed solution to a system of linear
   equations and provides error bounds and backward error estimates
   for the solution.  In addition to normwise error bound, the code
   provides maximum componentwise error bound if possible.  See
   comments for ERR_BNDS_NORM and ERR_BNDS_COMP for details of the
   error bounds.

   The original system of linear equations may have been equilibrated
   before calling this routine, as described by arguments EQUED, R
   and C below. In this case, the solution and error bounds returned
   are for the original unequilibrated system.                    Some␣
↪optional parameters are bundled in the PARAMS array.  These
    settings determine how refinement is performed, but often the
    defaults are acceptable.  If the defaults are acceptable, users
    can pass NPARAMS = 0 which prevents the source code from accessing
    the PARAMS argument.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dgbrfsx(TRANS, EQUED, N, KL, KU, NRHS, AB, LDAB, AFB, LDAFB, IPIV,
                   R, C, B, LDB, X, LDX, RCOND, BERR, N_ERR_BNDS,
                   ERR_BNDS_NORM, ERR_BNDS_COMP, NPARAMS, PARAMS, WORK, IWORK,
                   INFO)
```

C specification:

```
#include "armpl.h"

void dgbrfsx_(const char *trans, const char *equed, const armpl_int_t *n,
              const armpl_int_t *kl, const armpl_int_t *ku,
              const armpl_int_t *nrhs, const double *ab,
              const armpl_int_t *ldab, const double *afb,
              const armpl_int_t *ldafb, const armpl_int_t *ipiv,
              const double *r, const double *c, const double *b,
              const armpl_int_t *ldb, double *x, const armpl_int_t *ldx,
              double *rcond, double *berr, const armpl_int_t *n_err_bnds,
              double *err_bnds_norm, double *err_bnds_comp,
              const armpl_int_t *nparams, double *params, double *work,
              armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose = Transpose)

EQUED Input parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done to A before calling this routine. This is needed to compute the solution and error bounds correctly. = 'N': No equilibration = 'R': Row equilibration, i.e., A has been

premultiplied by diag(R). = 'C': Column equilibration, i.e., A has been postmultiplied by diag(C). = 'B': Both row and column equilibration, i.e., A has been replaced by diag(R) * A * diag(C). The right hand side B has been changed accordingly.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AB Input parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). The original band matrix A, stored in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(ku+1+i-j,j) = A(i,j) for max(1,j-ku)<=i<=min(n,j+kl).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KL+KU+1.

AFB Input parameter.

AFB is DOUBLE PRECISION

AFB is an array, dimension (LDAFB, N). Details of the LU factorization of the band matrix A, as computed by DGBTRF. U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1.

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AFB. LDAFB >= 2*KL*KU+1.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from DGETRF; for 1<=i<=N, row i of the matrix was interchanged with row IPIV(i).

R Input and output parameter.

R is DOUBLE PRECISION

R is an array, dimension (N). The row scale factors for A. If EQUED = 'R' or 'B', A is multiplied on the left by diag(R); if EQUED = 'N' or 'C', R is not accessed. R is an input argument if FACT = 'F'; otherwise, R is an output argument. If FACT = 'F' and EQUED = 'R' or 'B', each element of R must be positive. If R is output, each element of R is a power of the radix. If R is input, each element of R should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). The column scale factors for A. If EQUED = 'C' or 'B', A is multiplied on the right by diag(C); if EQUED = 'N' or 'R', C is not accessed. C is an input argument if FACT = 'F'; otherwise, C is an output argument. If FACT = 'F' and EQUED = 'C' or 'B', each element of C must be positive. If C is output, each element of C is a power of the radix. If C is input, each element of C should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by DGETRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). Componentwise relative backward error. This is the componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

N_ERR_BNDS Input parameter.

N_ERR_BNDS is INTEGER

Number of error bounds to return for each right hand side and each type (normwise or componentwise). See ERR_BNDS_NORM and ERR_BNDS_COMP below.

ERR_BNDS_NORM Output parameter.

ERR_BNDS_NORM is DOUBLE PRECISION

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 “Trust/don’t trust” boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * dlamch(‘Epsilon’).

err = 2 “Guaranteed” error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * dlamch(‘Epsilon’). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * dlamch(‘Epsilon’) to determine if the error estimate is “guaranteed”. These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Output parameter.

ERR_BNDS_COMP is DOUBLE PRECISION

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:, N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 “Trust/don’t trust” boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * dlamch(‘Epsilon’).

err = 2 “Guaranteed” error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * dlamch(‘Epsilon’). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * dlamch(‘Epsilon’) to determine if the error estimate is “guaranteed”. These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

NPARAMS Input parameter.

NPARAMS is INTEGER

Specifies the number of parameters set in PARAMS. If .LE. 0, the PARAMS array is never referenced and default values are used.

PARAMS Input and output parameter.

PARAMS is DOUBLE PRECISION

PARAMS is an array, dimension (NPARAMS). Specifies algorithm parameters. If an entry is .LT. 0.0, then that entry will be filled with default value used for that parameter. Only positions up to NPARAMS are accessed; defaults are used for higher-numbered parameters.

PARAMS(LA_LINRX_ITREF_I = 1) : Whether to perform iterative refinement or not. Default: 1.0D+0 = 0.0 : No refinement is performed, and no error bounds are computed. = 1.0 : Use the double-precision refinement algorithm, possibly with doubled-single computations if the compilation environment does not support DOUBLE PRECISION. (other values are reserved for future use)

PARAMS(LA_LINRX_ITHRESH_I = 2) : Maximum number of residual computations allowed for refinement. Default: 10 Aggressive: Set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in err_bnds_norm and err_bnds_comp may no longer be trustworthy.

PARAMS(LA_LINRX_CWISE_I = 3) : Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Default: 1.0 (attempt componentwise convergence)

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (4*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. The solution to every right-hand side is guaranteed. < 0: If INFO = -i, the i-th argument had an illegal value > 0 and <= N: U(INFO, INFO) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+J: The solution corresponding to the Jth right-hand side is not guaranteed. The solutions corresponding to other right- hand sides K with K > J may not be guaranteed as well, but only the first such right-hand side is reported. If a small componentwise error is not requested (PARAMS(3) = 0.0) then the Jth right-hand side is the first with a normwise error bound that is not guaranteed (the smallest J such that ERR_BNDS_NORM(J,1) = 0.0). By default (PARAMS(3) = 1.0) the Jth right-hand side is the first with either a normwise or componentwise error bound that is not guaranteed (the smallest J such that either ERR_BNDS_NORM(J,1) = 0.0 or ERR_BNDS_COMP(J,1) = 0.0). See the definition of ERR_BNDS_NORM(:,1) and ERR_BNDS_COMP(:,1). To get information about all of the right-hand sides check ERR_BNDS_NORM or ERR_BNDS_COMP.

Related Information

For this routine in other precisions, please see *cgbrfsx*, *sgbrfsx* and *zgbrfsx*. It also exists with a native C interface as *LAPACKE_dgbrfsx*.

4.9.22 dgerfs

dgerfs improves the computed solution to a system of linear equations and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgerfs(TRANS, N, NRHS, A, LDA, AF, LDAF, IPIV, B, LDB, X, LDX,
                  FERR, BERR, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgerfs_(const char *trans, const armpl_int_t *n, const armpl_int_t *nrhs,
             const double *a, const armpl_int_t *lda, const double *af,
             const armpl_int_t *ldaf, const armpl_int_t *ipiv,
             const double *b, const armpl_int_t *ldb, double *x,
             const armpl_int_t *ldx, double *ferr, double *berr, double *work,
             armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose = Transpose)

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). The original N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is DOUBLE PRECISION

AF is an array, dimension (LDAF, N). The factors L and U from the factorization A = P*L*U as computed by DGETRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from DGETRF; for 1<=i<=N, row i of the matrix was interchanged with row IPIV(i).

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by DGETRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgerfs*, *sgerfs* and *zgerfs*. It also exists with a native C interface as *LAPACKE_dgerfs*.

4.9.23 dgerfsx

```
DGERFSX improves the computed solution to a system of linear
   equations and provides error bounds and backward error estimates
   for the solution.  In addition to normwise error bound, the code
   provides maximum componentwise error bound if possible.  See
   comments for ERR_BNDS_NORM and ERR_BNDS_COMP for details of the
   error bounds.

   The original system of linear equations may have been equilibrated
   before calling this routine, as described by arguments EQUED, R
   and C below. In this case, the solution and error bounds returned
   are for the original unequilibrated system.                    Some␣
↪optional parameters are bundled in the PARAMS array.  These
```

(continues on next page)

(continued from previous page)

```
    settings determine how refinement is performed, but often the
    defaults are acceptable.  If the defaults are acceptable, users
    can pass NPARAMS = 0 which prevents the source code from accessing
    the PARAMS argument.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dgerfsx(TRANS, EQUED, N, NRHS, A, LDA, AF, LDAF, IPIV, R, C, B,
                   LDB, X, LDX, RCOND, BERR, N_ERR_BNDS, ERR_BNDS_NORM,
                   ERR_BNDS_COMP, NPARAMS, PARAMS, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgerfsx_(const char *trans, const char *equed, const armpl_int_t *n,
              const armpl_int_t *nrhs, const double *a,
              const armpl_int_t *lda, const double *af,
              const armpl_int_t *ldaf, const armpl_int_t *ipiv,
              const double *r, const double *c, const double *b,
              const armpl_int_t *ldb, double *x, const armpl_int_t *ldx,
              double *rcond, double *berr, const armpl_int_t *n_err_bnds,
              double *err_bnds_norm, double *err_bnds_comp,
              const armpl_int_t *nparams, double *params, double *work,
              armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose = Transpose)

EQUED Input parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done to A before calling this routine. This is needed to compute the solution and error bounds correctly. = 'N': No equilibration = 'R': Row equilibration, i.e., A has been premultiplied by diag(R). = 'C': Column equilibration, i.e., A has been postmultiplied by diag(C). = 'B': Both row and column equilibration, i.e., A has been replaced by diag(R) * A * diag(C). The right hand side B has been changed accordingly.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). The original N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is DOUBLE PRECISION

AF is an array, dimension (LDAF, N). The factors L and U from the factorization A = P*L*U as computed by DGETRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from DGETRF; for 1<=i<=N, row i of the matrix was interchanged with row IPIV(i).

R Input parameter.

R is DOUBLE PRECISION

R is an array, dimension (N). The row scale factors for A. If EQUED = 'R' or 'B', A is multiplied on the left by diag(R); if EQUED = 'N' or 'C', R is not accessed. If R is accessed, each element of R should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

C Input parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). The column scale factors for A. If EQUED = 'C' or 'B', A is multiplied on the right by diag(C); if EQUED = 'N' or 'R', C is not accessed. If C is accessed, each element of C should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by DGETRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). Componentwise relative backward error. This is the componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

N_ERR_BNDS Input parameter.

N_ERR_BNDS is INTEGER

Number of error bounds to return for each right hand side and each type (normwise or componentwise). See ERR_BNDS_NORM and ERR_BNDS_COMP below.

ERR_BNDS_NORM Output parameter.

ERR_BNDS_NORM is DOUBLE PRECISION

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 “Trust/don’t trust” boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * dlamch(‘Epsilon’).

err = 2 “Guaranteed” error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * dlamch(‘Epsilon’). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * dlamch(‘Epsilon’) to determine if the error estimate is “guaranteed”. These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Output parameter.

ERR_BNDS_COMP is DOUBLE PRECISION

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:, N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * dlamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * dlamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * dlamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

NPARAMS Input parameter.

NPARAMS is INTEGER

Specifies the number of parameters set in PARAMS. If .LE. 0, the PARAMS array is never referenced and default values are used.

PARAMS Input and output parameter.

PARAMS is DOUBLE PRECISION

PARAMS is an array, dimension (NPARAMS). Specifies algorithm parameters. If an entry is .LT. 0.0, then that entry will be filled with default value used for that parameter. Only positions up to NPARAMS are accessed; defaults are used for higher-numbered parameters.

PARAMS(LA_LINRX_ITREF_I = 1) : Whether to perform iterative refinement or not. Default: 1.0D+0 = 0.0 : No refinement is performed, and no error bounds are computed. = 1.0 : Use the double-precision refinement algorithm, possibly with doubled-single computations if the compilation environment does not support DOUBLE PRECISION. (other values are reserved for future use)

PARAMS(LA_LINRX_ITHRESH_I = 2) : Maximum number of residual computations allowed for refinement. Default: 10 Aggressive: Set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in err_bnds_norm and err_bnds_comp may no longer be trustworthy.

PARAMS(LA_LINRX_CWISE_I = 3) : Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Default: 1.0 (attempt componentwise convergence)

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (4*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. The solution to every right-hand side is guaranteed. < 0: If INFO = -i, the i-th argument had an illegal value > 0 and <= N: U(INFO, INFO) is exactly zero. The factorization has been

completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+J: The solution corresponding to the Jth right-hand side is not guaranteed. The solutions corresponding to other right- hand sides K with K > J may not be guaranteed as well, but only the first such right-hand side is reported. If a small componentwise error is not requested (PARAMS(3) = 0.0) then the Jth right-hand side is the first with a normwise error bound that is not guaranteed (the smallest J such that ERR_BNDS_NORM(J,1) = 0.0). By default (PARAMS(3) = 1.0) the Jth right-hand side is the first with either a normwise or componentwise error bound that is not guaranteed (the smallest J such that either ERR_BNDS_NORM(J,1) = 0.0 or ERR_BNDS_COMP(J,1) = 0.0). See the definition of ERR_BNDS_NORM(:,1) and ERR_BNDS_COMP(:,1). To get information about all of the right-hand sides check ERR_BNDS_NORM or ERR_BNDS_COMP.

Related Information

For this routine in other precisions, please see *cgerfsx*, *sgerfsx* and *zgerfsx*. It also exists with a native C interface as *LAPACKE_dgerfsx*.

4.9.24 dgtrfs

dgtrfs improves the computed solution to a system of linear equations when the coefficient matrix is tridiagonal, and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgtrfs(TRANS, N, NRHS, DL, D, DU, DLF, DF, DUF, DU2, IPIV, B, LDB,
                  X, LDX, FERR, BERR, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgtrfs_(const char *trans, const armpl_int_t *n, const armpl_int_t *nrhs,
             const double *dl, const double *d, const double *du,
             const double *dlf, const double *df, const double *duf,
             const double *du2, const armpl_int_t *ipiv, const double *b,
             const armpl_int_t *ldb, double *x, const armpl_int_t *ldx,
             double *ferr, double *berr, double *work, armpl_int_t *iwork,
             armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose = Transpose)

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

DL Input parameter.

DL is DOUBLE PRECISION

DL is an array, dimension (N-1). The (n-1) subdiagonal elements of A.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The diagonal elements of A.

DU Input parameter.

DU is DOUBLE PRECISION

DU is an array, dimension (N-1). The (n-1) superdiagonal elements of A.

DLF Input parameter.

DLF is DOUBLE PRECISION

DLF is an array, dimension (N-1). The (n-1) multipliers that define the matrix L from the LU factorization of A as computed by DGTTRF.

DF Input parameter.

DF is DOUBLE PRECISION

DF is an array, dimension (N). The n diagonal elements of the upper triangular matrix U from the LU factorization of A.

DUF Input parameter.

DUF is DOUBLE PRECISION

DUF is an array, dimension (N-1). The (n-1) elements of the first superdiagonal of U.

DU2 Input parameter.

DU2 is DOUBLE PRECISION

DU2 is an array, dimension (N-2). The (n-2) elements of the second superdiagonal of U.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices; for 1 <= i <= n, row i of the matrix was interchanged with row IPIV(i). IPIV(i) will always be either i or i+1; IPIV(i) = i indicates a row interchange was not required.

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by DGTTRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgtrfs*, *sgtrfs* and *zgtrfs*. It also exists with a native C interface as *LAPACKE_dgtrfs*.

4.9.25 dpbrfs

dpbrfs improves the computed solution to a system of linear equations when the coefficient matrix is symmetric positive definite and banded, and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine dpbrfs(UPLO, N, KD, NRHS, AB, LDAB, AFB, LDAFB, B, LDB, X, LDX,
                  FERR, BERR, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dpbrfs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
             const armpl_int_t *nrhs, const double *ab,
```

(continues on next page)

(continued from previous page)

```
                const armpl_int_t *ldab, const double *afb,
                const armpl_int_t *ldafb, const double *b,
                const armpl_int_t *ldb, double *x, const armpl_int_t *ldx,
                double *ferr, double *berr, double *work, armpl_int_t *iwork,
                armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AB Input parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). The upper or lower triangle of the symmetric band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

AFB Input parameter.

AFB is DOUBLE PRECISION

AFB is an array, dimension (LDAFB, N). The triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T of the band matrix A as computed by DPBTRF, in the same storage format as A (see AB).

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AFB. LDAFB >= KD+1.

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by DPBTRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cpbrfs*, *spbrfs* and *zpbrfs*. It also exists with a native C interface as *LAPACKE_dpbrfs*.

4.9.26 dporfs

dporfs improves the computed solution to a system of linear equations when the coefficient matrix is symmetric positive definite, and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine dporfs(UPLO, N, NRHS, A, LDA, AF, LDAF, B, LDB, X, LDX, FERR, BERR,
                  WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dporfs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const double *a, const armpl_int_t *lda, const double *af,
             const armpl_int_t *ldaf, const double *b, const armpl_int_t *ldb,
             double *x, const armpl_int_t *ldx, double *ferr, double *berr,
             double *work, armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). The symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is DOUBLE PRECISION

AF is an array, dimension (LDAF, N). The triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , as computed by DPOTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by DPOTRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cporfs*, *sporfs* and *zporfs*. It also exists with a native C interface as *LAPACKE_dporfs*.

4.9.27 dporfsx

```
DPORFSX improves the computed solution to a system of linear
   equations when the coefficient matrix is symmetric positive
   definite, and provides error bounds and backward error estimates
   for the solution.  In addition to normwise error bound, the code
   provides maximum componentwise error bound if possible.  See
   comments for ERR_BNDS_NORM and ERR_BNDS_COMP for details of the
   error bounds.

   The original system of linear equations may have been equilibrated
   before calling this routine, as described by arguments EQUED and S
   below. In this case, the solution and error bounds returned are
   for the original unequilibrated system.                    Some optional␣
↪parameters are bundled in the PARAMS array.  These
    settings determine how refinement is performed, but often the
    defaults are acceptable.  If the defaults are acceptable, users
    can pass NPARAMS = 0 which prevents the source code from accessing
    the PARAMS argument.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dporfsx(UPLO, EQUED, N, NRHS, A, LDA, AF, LDAF, S, B, LDB, X, LDX,
                   RCOND, BERR, N_ERR_BNDS, ERR_BNDS_NORM, ERR_BNDS_COMP,
                   NPARAMS, PARAMS, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dporfsx_(const char *uplo, const char *equed, const armpl_int_t *n,
              const armpl_int_t *nrhs, const double *a,
              const armpl_int_t *lda, const double *af,
              const armpl_int_t *ldaf, const double *s, const double *b,
              const armpl_int_t *ldb, double *x, const armpl_int_t *ldx,
              double *rcond, double *berr, const armpl_int_t *n_err_bnds,
              double *err_bnds_norm, double *err_bnds_comp,
              const armpl_int_t *nparams, double *params, double *work,
              armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

EQUED Input parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done to A before calling this routine. This is needed to compute the solution and error bounds correctly. = 'N': No equilibration = 'Y': Both row and column equilibration, i.e., A has been replaced by diag(S) * A * diag(S). The right hand side B has been changed accordingly.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). The symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is DOUBLE PRECISION

AF is an array, dimension (LDAF, N). The triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , as computed by DPOTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

S Input and output parameter.

S is DOUBLE PRECISION

S is an array, dimension (N). The row scale factors for A. If EQUED = 'Y', A is multiplied on the left and right by diag(S). S is an input argument if FACT = 'F'; otherwise, S is an output argument. If FACT = 'F' and EQUED = 'Y', each element of S must be positive. If S is output, each element of S is a power of the radix. If S is input, each element of S should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by DGETRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). Componentwise relative backward error. This is the componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

N_ERR_BNDS Input parameter.

N_ERR_BNDS is INTEGER

Number of error bounds to return for each right hand side and each type (normwise or componentwise). See ERR_BNDS_NORM and ERR_BNDS_COMP below.

ERR_BNDS_NORM Output parameter.

ERR_BNDS_NORM is DOUBLE PRECISION

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 “Trust/don’t trust” boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * dlamch(‘Epsilon’).

err = 2 “Guaranteed” error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * dlamch(‘Epsilon’). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * dlamch(‘Epsilon’) to determine if the error estimate is “guaranteed”. These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Output parameter.

ERR_BNDS_COMP is DOUBLE PRECISION

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then

ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:, N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * dlamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * dlamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * dlamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

NPARAMS Input parameter.

NPARAMS is INTEGER

Specifies the number of parameters set in PARAMS. If .LE. 0, the PARAMS array is never referenced and default values are used.

PARAMS Input and output parameter.

PARAMS is DOUBLE PRECISION

PARAMS is an array, dimension (NPARAMS). Specifies algorithm parameters. If an entry is .LT. 0.0, then that entry will be filled with default value used for that parameter. Only positions up to NPARAMS are accessed; defaults are used for higher-numbered parameters.

PARAMS(LA_LINRX_ITREF_I = 1) : Whether to perform iterative refinement or not. Default: 1.0D+0 = 0.0 : No refinement is performed, and no error bounds are computed. = 1.0 : Use the double-precision refinement algorithm, possibly with doubled-single computations if the compilation environment does not support DOUBLE PRECISION. (other values are reserved for future use)

PARAMS(LA_LINRX_ITHRESH_I = 2) : Maximum number of residual computations allowed for refinement. Default: 10 Aggressive: Set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in err_bnds_norm and err_bnds_comp may no longer be trustworthy.

PARAMS(LA_LINRX_CWISE_I = 3) : Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Default: 1.0 (attempt componentwise convergence)

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (4*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. The solution to every right-hand side is guaranteed. < 0: If INFO = -i, the i-th argument had an illegal value > 0 and <= N: U(INFO, INFO) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+J: The solution corresponding to the Jth right-hand side is not guaranteed. The solutions corresponding to other right- hand sides K with K > J may not be guaranteed as well, but only

the first such right-hand side is reported. If a small componentwise error is not requested (PARAMS(3) = 0.0) then the Jth right-hand side is the first with a normwise error bound that is not guaranteed (the smallest J such that ERR_BNDS_NORM(J,1) = 0.0). By default (PARAMS(3) = 1.0) the Jth right-hand side is the first with either a normwise or componentwise error bound that is not guaranteed (the smallest J such that either ERR_BNDS_NORM(J,1) = 0.0 or ERR_BNDS_COMP(J,1) = 0.0). See the definition of ERR_BNDS_NORM(:,1) and ERR_BNDS_COMP(:,1). To get information about all of the right-hand sides check `ERR_BNDS_NORM` or `ERR_BNDS_COMP`.

Related Information

For this routine in other precisions, please see *cporfsx*, *sporfsx* and *zporfsx*. It also exists with a native C interface as *LAPACKE_dporfsx*.

4.9.28 dpprfs

`dpprfs` improves the computed solution to a system of linear equations when the coefficient matrix is symmetric positive definite and packed, and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine dpprfs(UPLO, N, NRHS, AP, AFP, B, LDB, X, LDX, FERR, BERR, WORK,
                  IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dpprfs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const double *ap, const double *afp, const double *b,
             const armpl_int_t *ldb, double *x, const armpl_int_t *ldx,
             double *ferr, double *berr, double *work, armpl_int_t *iwork,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

= '`U`': Upper triangle of A is stored; = '`L`': Lower triangle of A is stored.

N Input parameter.

`N` is `INTEGER`

The order of the matrix A. `N` >= 0.

NRHS Input parameter.

`NRHS` is `INTEGER`

The number of right hand sides, i.e., the number of columns of the matrices `B` and `X`. `NRHS` >= 0.

AP Input parameter.

`AP` is `DOUBLE PRECISION`

AP is an array, dimension (N*(N+1)/2). The upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

AFP Input parameter.

AFP is DOUBLE PRECISION

AFP is an array, dimension (N*(N+1)/2). The triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , as computed by DPPTRF/ZPPTRF, packed columnwise in a linear array in the same format as A (see AP).

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by DPPTRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cpprfs*, *spprfs* and *zpprfs*. It also exists with a native C interface as *LAPACKE_dpprfs*.

4.9.29 dptrfs

`dptrfs` improves the computed solution to a system of linear equations when the coefficient matrix is symmetric positive definite and tridiagonal, and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine dptrfs(N, NRHS, D, E, DF, EF, B, LDB, X, LDX, FERR, BERR, WORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dptrfs_(const armpl_int_t *n, const armpl_int_t *nrhs, const double *d,
             const double *e, const double *df, const double *ef,
             const double *b, const armpl_int_t *ldb, double *x,
             const armpl_int_t *ldx, double *ferr, double *berr, double *work,
             armpl_int_t *info);
```

Parameters

N Input parameter.

`N` is `INTEGER`

The order of the matrix A. `N` >= 0.

NRHS Input parameter.

`NRHS` is `INTEGER`

The number of right hand sides, i.e., the number of columns of the matrix `B`. `NRHS` >= 0.

D Input parameter.

`D` is `DOUBLE PRECISION`

`D` is an array, dimension (`N`). The n diagonal elements of the tridiagonal matrix A.

E Input parameter.

`E` is `DOUBLE PRECISION`

`E` is an array, dimension (N-1). The (n-1) subdiagonal elements of the tridiagonal matrix A.

DF Input parameter.

`DF` is `DOUBLE PRECISION`

`DF` is an array, dimension (`N`). The n diagonal elements of the diagonal matrix `D` from the factorization computed by DPTTRF.

EF Input parameter.

EF is DOUBLE PRECISION

EF is an array, dimension (N-1). The (n-1) subdiagonal elements of the unit bidiagonal factor L from the factorization computed by DPTTRF.

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by DPTTRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j).

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cptrfs*, *sptrfs* and *zptrfs*. It also exists with a native C interface as *LAPACKE_dptrfs*.

4.9.30 dsprfs

dsprfs improves the computed solution to a system of linear equations when the coefficient matrix is symmetric indefinite and packed, and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsprfs(UPLO, N, NRHS, AP, AFP, IPIV, B, LDB, X, LDX, FERR, BERR,
                  WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dsprfs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const double *ap, const double *afp, const armpl_int_t *ipiv,
             const double *b, const armpl_int_t *ldb, double *x,
             const armpl_int_t *ldx, double *ferr, double *berr, double *work,
             armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AP Input parameter.

AP is DOUBLE PRECISION

AP is an array, dimension (N*(N+1)/2). The upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n.

AFP Input parameter.

AFP is DOUBLE PRECISION

AFP is an array, dimension (N*(N+1)/2). The factored form of the matrix A. AFP contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by DSPTRF, stored as a packed triangular matrix.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by DSPTRF.

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by DSPTRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csprfs*, *ssprfs* and *zsprfs*. It also exists with a native C interface as *LAPACKE_dsprfs*.

4.9.31 dsyrfs

dsyrfs improves the computed solution to a system of linear equations when the coefficient matrix is symmetric indefinite, and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsyrfs(UPLO, N, NRHS, A, LDA, AF, LDAF, IPIV, B, LDB, X, LDX, FERR,
                  BERR, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dsyrfs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const double *a, const armpl_int_t *lda, const double *af,
             const armpl_int_t *ldaf, const armpl_int_t *ipiv,
             const double *b, const armpl_int_t *ldb, double *x,
             const armpl_int_t *ldx, double *ferr, double *berr, double *work,
             armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). The symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is DOUBLE PRECISION

AF is an array, dimension (LDAF, N). The factored form of the matrix A. AF contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by DSYTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by DSYTRF.

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by DSYTRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csyrfs*, *ssyrfs* and *zsyrfs*. It also exists with a native C interface as *LAPACKE_dsyrfs*.

4.9.32 dsyrfsx

```
DSYRFSX improves the computed solution to a system of linear
   equations when the coefficient matrix is symmetric indefinite, and
   provides error bounds and backward error estimates for the
   solution.  In addition to normwise error bound, the code provides
   maximum componentwise error bound if possible.  See comments for
   ERR_BNDS_NORM and ERR_BNDS_COMP for details of the error bounds.

   The original system of linear equations may have been equilibrated
   before calling this routine, as described by arguments EQUED and S
   below. In this case, the solution and error bounds returned are
   for the original unequilibrated system.                    Some optional␣
↪parameters are bundled in the PARAMS array.  These
    settings determine how refinement is performed, but often the
    defaults are acceptable.  If the defaults are acceptable, users
    can pass NPARAMS = 0 which prevents the source code from accessing
    the PARAMS argument.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dsyrfsx(UPLO, EQUED, N, NRHS, A, LDA, AF, LDAF, IPIV, S, B, LDB, X,
                   LDX, RCOND, BERR, N_ERR_BNDS, ERR_BNDS_NORM, ERR_BNDS_COMP,
                   NPARAMS, PARAMS, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dsyrfsx_(const char *uplo, const char *equed, const armpl_int_t *n,
              const armpl_int_t *nrhs, const double *a,
              const armpl_int_t *lda, const double *af,
              const armpl_int_t *ldaf, const armpl_int_t *ipiv,
              const double *s, const double *b, const armpl_int_t *ldb,
              double *x, const armpl_int_t *ldx, double *rcond, double *berr,
              const armpl_int_t *n_err_bnds, double *err_bnds_norm,
              double *err_bnds_comp, const armpl_int_t *nparams,
              double *params, double *work, armpl_int_t *iwork,
              armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

EQUED Input parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done to A before calling this routine. This is needed to compute the solution and error bounds correctly. = 'N': No equilibration = 'Y': Both row and column equilibration, i.e., A has been replaced by diag(S) * A * diag(S). The right hand side B has been changed accordingly.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). The symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is DOUBLE PRECISION

AF is an array, dimension (LDAF, N). The factored form of the matrix A. AF contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by DSYTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by DSYTRF.

S Input and output parameter.

S is DOUBLE PRECISION

S is an array, dimension (N). The scale factors for A. If EQUED = 'Y', A is multiplied on the left and right by diag(S). S is an input argument if FACT = 'F'; otherwise, S is an output argument. If FACT = 'F' and EQUED = 'Y', each element of S must be positive. If S is output, each element of S is a power of the radix. If S is input, each element of S should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by DGETRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). Componentwise relative backward error. This is the componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

N_ERR_BNDS Input parameter.

N_ERR_BNDS is INTEGER

Number of error bounds to return for each right hand side and each type (normwise or componentwise). See ERR_BNDS_NORM and ERR_BNDS_COMP below.

ERR_BNDS_NORM Output parameter.

ERR_BNDS_NORM is DOUBLE PRECISION

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 “Trust/don’t trust” boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * dlamch(‘Epsilon’).

err = 2 “Guaranteed” error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * dlamch(‘Epsilon’). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * dlamch(‘Epsilon’) to determine if the error estimate is “guaranteed”. These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Output parameter.

ERR_BNDS_COMP is DOUBLE PRECISION

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:, N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * dlamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * dlamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * dlamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

NPARAMS Input parameter.

NPARAMS is INTEGER

Specifies the number of parameters set in PARAMS. If .LE. 0, the PARAMS array is never referenced and default values are used.

PARAMS Input and output parameter.

PARAMS is DOUBLE PRECISION

PARAMS is an array, dimension (NPARAMS). Specifies algorithm parameters. If an entry is .LT. 0.0, then that entry will be filled with default value used for that parameter. Only positions up to NPARAMS are accessed; defaults are used for higher-numbered parameters.

PARAMS(LA_LINRX_ITREF_I = 1) : Whether to perform iterative refinement or not. Default: 1.0D+0 = 0.0 : No refinement is performed, and no error bounds are computed. = 1.0 : Use the double-precision refinement algorithm, possibly with doubled-single computations if the compilation environment does not support DOUBLE PRECISION. (other values are reserved for future use)

PARAMS(LA_LINRX_ITHRESH_I = 2) : Maximum number of residual computations allowed for refinement. Default: 10 Aggressive: Set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in err_bnds_norm and err_bnds_comp may no longer be trustworthy.

PARAMS(LA_LINRX_CWISE_I = 3) : Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Default: 1.0 (attempt componentwise convergence)

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (4*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. The solution to every right-hand side is guaranteed. < 0: If INFO = -i, the i-th argument had an illegal value > 0 and <= N: U(INFO, INFO) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+J: The solution corresponding to the Jth right-hand side is not guaranteed. The solutions corresponding to other right- hand sides K with K > J may not be guaranteed as well, but only the first such right-hand side is reported. If a small componentwise error is not requested (PARAMS(3) = 0.0) then the Jth right-hand side is the first with a normwise error bound that is not guaranteed (the smallest J such that ERR_BNDS_NORM(J,1) = 0.0). By default (PARAMS(3) = 1.0) the Jth right-hand side is the first with either a normwise or componentwise error bound that is not guaranteed (the smallest J such that either ERR_BNDS_NORM(J,1) = 0.0 or ERR_BNDS_COMP(J,1) = 0.0). See the definition of ERR_BNDS_NORM(:,1) and ERR_BNDS_COMP(:,1). To get information about all of the right-hand sides check ERR_BNDS_NORM or ERR_BNDS_COMP.

Related Information

For this routine in other precisions, please see *csyrfsx*, *ssyrfsx* and *zsyrfsx*. It also exists with a native C interface as *LAPACKE_dsyrfsx*.

4.9.33 dtbrfs

dtbrfs provides error bounds and backward error estimates for the solution to a system of linear equations with a triangular band coefficient matrix.

The solution matrix X must be computed by DTBTRS or some other means before entering this routine. dtbrfs does not do iterative refinement because doing so cannot improve the backward error.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtbrfs(UPLO, TRANS, DIAG, N, KD, NRHS, AB, LDAB, B, LDB, X, LDX,
                  FERR, BERR, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dtbrfs_(const char *uplo, const char *trans, const char *diag,
             const armpl_int_t *n, const armpl_int_t *kd,
             const armpl_int_t *nrhs, const double *ab,
             const armpl_int_t *ldab, const double *b, const armpl_int_t *ldb,
             const double *x, const armpl_int_t *ldx, double *ferr,
             double *berr, double *work, armpl_int_t *iwork,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose = Transpose)

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals or subdiagonals of the triangular band matrix A. KD >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AB Input parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). The upper or lower triangular band matrix A, stored in the first kd+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd). If DIAG = 'U', the diagonal elements of A are not referenced and are assumed to be 1.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, NRHS). The solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctbrfs*, *stbrfs* and *ztbrfs*. It also exists with a native C interface as *LAPACKE_dtbrfs*.

4.9.34 dtprfs

dtprfs provides error bounds and backward error estimates for the solution to a system of linear equations with a triangular packed coefficient matrix.

The solution matrix X must be computed by DTPTRS or some other means before entering this routine. dtprfs does not do iterative refinement because doing so cannot improve the backward error.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtprfs(UPLO, TRANS, DIAG, N, NRHS, AP, B, LDB, X, LDX, FERR, BERR,
                  WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dtprfs_(const char *uplo, const char *trans, const char *diag,
             const armpl_int_t *n, const armpl_int_t *nrhs, const double *ap,
             const double *b, const armpl_int_t *ldb, const double *x,
             const armpl_int_t *ldx, double *ferr, double *berr, double *work,
             armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose = Transpose)

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AP Input parameter.

AP is DOUBLE PRECISION

AP is an array, dimension (N*(N+1)/2). The upper or lower triangular matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n. If DIAG = 'U', the diagonal elements of A are not referenced and are assumed to be 1.

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, NRHS). The solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctprfs*, *stprfs* and *ztprfs*. It also exists with a native C interface as *LAPACKE_dtprfs*.

4.9.35 dtrrfs

dtrrfs provides error bounds and backward error estimates for the solution to a system of linear equations with a triangular coefficient matrix.

The solution matrix X must be computed by DTRTRS or some other means before entering this routine. dtrrfs does not do iterative refinement because doing so cannot improve the backward error.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtrrfs(UPLO, TRANS, DIAG, N, NRHS, A, LDA, B, LDB, X, LDX, FERR,
                  BERR, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dtrrfs_(const char *uplo, const char *trans, const char *diag,
             const armpl_int_t *n, const armpl_int_t *nrhs, const double *a,
             const armpl_int_t *lda, const double *b, const armpl_int_t *ldb,
             const double *x, const armpl_int_t *ldx, double *ferr,
             double *berr, double *work, armpl_int_t *iwork,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose = Transpose)

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). The triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, NRHS). The solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctrrfs*, *strrfs* and *ztrrfs*. It also exists with a native C interface as *LAPACKE_dtrrfs*.

4.9.36 sgbrfs

sgbrfs improves the computed solution to a system of linear equations when the coefficient matrix is banded, and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgbrfs(TRANS, N, KL, KU, NRHS, AB, LDAB, AFB, LDAFB, IPIV, B, LDB,
                  X, LDX, FERR, BERR, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgbrfs_(const char *trans, const armpl_int_t *n, const armpl_int_t *kl,
             const armpl_int_t *ku, const armpl_int_t *nrhs, const float *ab,
             const armpl_int_t *ldab, const float *afb,
             const armpl_int_t *ldafb, const armpl_int_t *ipiv,
             const float *b, const armpl_int_t *ldb, float *x,
             const armpl_int_t *ldx, float *ferr, float *berr, float *work,
             armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose = Transpose)

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AB Input parameter.

AB is REAL

AB is an array, dimension (LDAB, N). The original band matrix A, stored in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(ku+1+i-j,j) = A(i,j) for max(1,j-ku)<=i<=min(n,j+kl).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KL+KU+1.

AFB Input parameter.

AFB is REAL

AFB is an array, dimension (LDAFB, N). Details of the LU factorization of the band matrix A, as computed by SGBTRF. U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1.

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AFB. LDAFB >= 2*KL*KU+1.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from SGBTRF; for 1<=i<=N, row i of the matrix was interchanged with row IPIV(i).

B Input parameter.

B is REAL

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is REAL

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by SGBTRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgbrfs*, *dgbrfs* and *zgbrfs*. It also exists with a native C interface as *LAPACKE_sgbrfs*.

4.9.37 sgbrfsx

```
SGBRFSX improves the computed solution to a system of linear
   equations and provides error bounds and backward error estimates
   for the solution.  In addition to normwise error bound, the code
   provides maximum componentwise error bound if possible.  See
   comments for ERR_BNDS_NORM and ERR_BNDS_COMP for details of the
   error bounds.

   The original system of linear equations may have been equilibrated
   before calling this routine, as described by arguments EQUED, R
   and C below. In this case, the solution and error bounds returned
   are for the original unequilibrated system.                    Some␣
↪optional parameters are bundled in the PARAMS array.  These
    settings determine how refinement is performed, but often the
    defaults are acceptable.  If the defaults are acceptable, users
    can pass NPARAMS = 0 which prevents the source code from accessing
    the PARAMS argument.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine sgbrfsx(TRANS, EQUED, N, KL, KU, NRHS, AB, LDAB, AFB, LDAFB, IPIV,
                   R, C, B, LDB, X, LDX, RCOND, BERR, N_ERR_BNDS,
                   ERR_BNDS_NORM, ERR_BNDS_COMP, NPARAMS, PARAMS, WORK, IWORK,
                   INFO)
```

C specification:

```
#include "armpl.h"

void sgbrfsx_(const char *trans, const char *equed, const armpl_int_t *n,
              const armpl_int_t *kl, const armpl_int_t *ku,
              const armpl_int_t *nrhs, const float *ab,
              const armpl_int_t *ldab, const float *afb,
              const armpl_int_t *ldafb, const armpl_int_t *ipiv,
              const float *r, const float *c, const float *b,
              const armpl_int_t *ldb, float *x, const armpl_int_t *ldx,
              float *rcond, float *berr, const armpl_int_t *n_err_bnds,
              float *err_bnds_norm, float *err_bnds_comp,
              const armpl_int_t *nparams, float *params, float *work,
              armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose = Transpose)

EQUED Input parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done to A before calling this routine. This is needed to compute the solution and error bounds correctly. = 'N': No equilibration = 'R': Row equilibration, i.e., A has been premultiplied by diag(R). = 'C': Column equilibration, i.e., A has been postmultiplied by diag(C). = 'B': Both row and column equilibration, i.e., A has been replaced by diag(R) * A * diag(C). The right hand side B has been changed accordingly.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AB Input parameter.

AB is REAL

AB is an array, dimension (LDAB, N). The original band matrix A, stored in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(ku+1+i-j,j) = A(i,j) for max(1,j-ku)<=i<=min(n,j+kl).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KL+KU+1.

AFB Input parameter.

AFB is REAL

AFB is an array, dimension (LDAFB, N). Details of the LU factorization of the band matrix A, as computed by DGBTRF. U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1.

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AFB. LDAFB >= 2*KL*KU+1.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from SGETRF; for 1<=i<=N, row i of the matrix was interchanged with row IPIV(i).

R Input and output parameter.

R is REAL

R is an array, dimension (N). The row scale factors for A. If EQUED = 'R' or 'B', A is multiplied on the left by diag(R); if EQUED = 'N' or 'C', R is not accessed. R is an input argument if FACT = 'F'; otherwise, R is an output argument. If FACT = 'F' and EQUED = 'R' or 'B', each element of R must be positive. If R is output, each element of R is a power of the radix. If R is input, each element of R should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

C Input and output parameter.

C is REAL

C is an array, dimension (N). The column scale factors for A. If EQUED = 'C' or 'B', A is multiplied on the right by diag(C); if EQUED = 'N' or 'R', C is not accessed. C is an input argument if FACT = 'F'; otherwise, C is an output argument. If FACT = 'F' and EQUED = 'C' or 'B', each element of C must be positive. If C is output, each element of C is a power of the radix. If C is input, each element of C should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input parameter.

B is REAL

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is REAL

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by SGETRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). Componentwise relative backward error. This is the componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

N_ERR_BNDS Input parameter.

N_ERR_BNDS is INTEGER

Number of error bounds to return for each right hand side and each type (normwise or componentwise). See ERR_BNDS_NORM and ERR_BNDS_COMP below.

ERR_BNDS_NORM Output parameter.

ERR_BNDS_NORM is REAL

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 “Trust/don’t trust” boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch(‘Epsilon’).

err = 2 “Guaranteed” error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch(‘Epsilon’). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch(‘Epsilon’) to determine if the error estimate is “guaranteed”. These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Output parameter.

ERR_BNDS_COMP is REAL

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:, N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

NPARAMS Input parameter.

NPARAMS is INTEGER

Specifies the number of parameters set in PARAMS. If .LE. 0, the PARAMS array is never referenced and default values are used.

PARAMS Input and output parameter.

PARAMS is REAL

PARAMS is an array, dimension NPARAMS. Specifies algorithm parameters. If an entry is .LT. 0.0, then that entry will be filled with default value used for that parameter. Only positions up to NPARAMS are accessed; defaults are used for higher-numbered parameters.

PARAMS(LA_LINRX_ITREF_I = 1) : Whether to perform iterative refinement or not. Default: 1.0 = 0.0 : No refinement is performed, and no error bounds are computed. = 1.0 : Use the double-precision refinement algorithm, possibly with doubled-single computations if the compilation environment does not support DOUBLE PRECISION. (other values are reserved for future use)

PARAMS(LA_LINRX_ITHRESH_I = 2) : Maximum number of residual computations allowed for refinement. Default: 10 Aggressive: Set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in err_bnds_norm and err_bnds_comp may no longer be trustworthy.

PARAMS(LA_LINRX_CWISE_I = 3) : Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Default: 1.0 (attempt componentwise convergence)

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (4*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. The solution to every right-hand side is guaranteed. < 0: If INFO = -i, the i-th argument had an illegal value > 0 and <= N: U(INFO, INFO) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+J: The solution corresponding to the Jth right-hand side is not guaranteed. The solutions corresponding to other right- hand sides K with K > J may not be guaranteed as well, but only the first such right-hand side is reported. If a small componentwise error is not requested (PARAMS(3) = 0.0) then the Jth right-hand side is the first with a normwise error bound that is not guaranteed (the smallest J such that ERR_BNDS_NORM(J,1) = 0.0). By default (PARAMS(3) = 1.0) the Jth right-hand side is the first with either a normwise or componentwise error bound that is not guaranteed (the smallest J such that either ERR_BNDS_NORM(J,1) = 0.0 or ERR_BNDS_COMP(J,1) = 0.0). See the definition of ERR_BNDS_NORM(:,1) and ERR_BNDS_COMP(:,1). To get information about all of the right-hand sides check ERR_BNDS_NORM or ERR_BNDS_COMP.

Related Information

For this routine in other precisions, please see *cgbrfsx*, *dgbrfsx* and *zgbrfsx*. It also exists with a native C interface as *LAPACKE_sgbrfsx*.

4.9.38 sgerfs

sgerfs improves the computed solution to a system of linear equations and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgerfs(TRANS, N, NRHS, A, LDA, AF, LDAF, IPIV, B, LDB, X, LDX,
                  FERR, BERR, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgerfs_(const char *trans, const armpl_int_t *n, const armpl_int_t *nrhs,
             const float *a, const armpl_int_t *lda, const float *af,
             const armpl_int_t *ldaf, const armpl_int_t *ipiv, const float *b,
             const armpl_int_t *ldb, float *x, const armpl_int_t *ldx,
             float *ferr, float *berr, float *work, armpl_int_t *iwork,
             armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose = Transpose)

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). The original N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is REAL

AF is an array, dimension (LDAF, N). The factors L and U from the factorization A = P*L*U as computed by SGETRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from SGETRF; for 1<=i<=N, row i of the matrix was interchanged with row IPIV(i).

B Input parameter.

B is REAL

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is REAL

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by SGETRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgerfs*, *dgerfs* and *zgerfs*. It also exists with a native C interface as *LAPACKE_sgerfs*.

4.9.39 sgerfsx

```
SGERFSX improves the computed solution to a system of linear
   equations and provides error bounds and backward error estimates
   for the solution.  In addition to normwise error bound, the code
   provides maximum componentwise error bound if possible.  See
   comments for ERR_BNDS_NORM and ERR_BNDS_COMP for details of the
   error bounds.

   The original system of linear equations may have been equilibrated
   before calling this routine, as described by arguments EQUED, R
   and C below. In this case, the solution and error bounds returned
   are for the original unequilibrated system.                    Some␣
↪optional parameters are bundled in the PARAMS array.  These
    settings determine how refinement is performed, but often the
    defaults are acceptable.  If the defaults are acceptable, users
    can pass NPARAMS = 0 which prevents the source code from accessing
    the PARAMS argument.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine sgerfsx(TRANS, EQUED, N, NRHS, A, LDA, AF, LDAF, IPIV, R, C, B,
                   LDB, X, LDX, RCOND, BERR, N_ERR_BNDS, ERR_BNDS_NORM,
                   ERR_BNDS_COMP, NPARAMS, PARAMS, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgerfsx_(const char *trans, const char *equed, const armpl_int_t *n,
              const armpl_int_t *nrhs, const float *a, const armpl_int_t *lda,
              const float *af, const armpl_int_t *ldaf,
              const armpl_int_t *ipiv, const float *r, const float *c,
              const float *b, const armpl_int_t *ldb, float *x,
              const armpl_int_t *ldx, float *rcond, float *berr,
              const armpl_int_t *n_err_bnds, float *err_bnds_norm,
              float *err_bnds_comp, const armpl_int_t *nparams, float *params,
              float *work, armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose = Transpose)

EQUED Input parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done to A before calling this routine. This is needed to compute the solution and error bounds correctly. = 'N': No equilibration = 'R': Row equilibration, i.e., A has been premultiplied by diag(R). = 'C': Column equilibration, i.e., A has been postmultiplied by diag(C). = 'B': Both row and column equilibration, i.e., A has been replaced by diag(R) * A * diag(C). The right hand side B has been changed accordingly.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). The original N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is REAL

AF is an array, dimension (LDAF, N). The factors L and U from the factorization A = P*L*U as computed by SGETRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from SGETRF; for 1<=i<=N, row i of the matrix was interchanged with row IPIV(i).

R Input parameter.

R is REAL

R is an array, dimension (N). The row scale factors for A. If EQUED = 'R' or 'B', A is multiplied on the left by diag(R); if EQUED = 'N' or 'C', R is not accessed. If R is accessed, each element of R should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

C Input parameter.

C is REAL

C is an array, dimension (N). The column scale factors for A. If EQUED = 'C' or 'B', A is multiplied on the right by diag(C); if EQUED = 'N' or 'R', C is not accessed. If C is accessed, each element of C should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input parameter.

B is REAL

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is REAL

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by SGETRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). Componentwise relative backward error. This is the componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

N_ERR_BNDS Input parameter.

N_ERR_BNDS is INTEGER

Number of error bounds to return for each right hand side and each type (normwise or componentwise). See ERR_BNDS_NORM and ERR_BNDS_COMP below.

ERR_BNDS_NORM Output parameter.

ERR_BNDS_NORM is REAL

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Output parameter.

ERR_BNDS_COMP is REAL

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:, N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

NPARAMS Input parameter.

NPARAMS is INTEGER

Specifies the number of parameters set in PARAMS. If .LE. 0, the PARAMS array is never referenced and default values are used.

PARAMS Input and output parameter.

PARAMS is REAL

PARAMS is an array, dimension NPARAMS. Specifies algorithm parameters. If an entry is .LT. 0.0, then that entry will be filled with default value used for that parameter. Only positions up to NPARAMS are accessed; defaults are used for higher-numbered parameters.

PARAMS(LA_LINRX_ITREF_I = 1) : Whether to perform iterative refinement or not. Default: 1.0 = 0.0 : No refinement is performed, and no error bounds are computed. = 1.0 : Use the double-precision refinement algorithm, possibly with doubled-single computations if the compilation environment does not support DOUBLE PRECISION. (other values are reserved for future use)

PARAMS(LA_LINRX_ITHRESH_I = 2) : Maximum number of residual computations allowed for refinement. Default: 10 Aggressive: Set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in err_bnds_norm and err_bnds_comp may no longer be trustworthy.

PARAMS(LA_LINRX_CWISE_I = 3) : Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Default: 1.0 (attempt componentwise convergence)

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (4*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. The solution to every right-hand side is guaranteed. < 0: If INFO = -i, the i-th argument had an illegal value > 0 and <= N: U(INFO, INFO) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+J: The solution corresponding to the Jth right-hand side is not guaranteed. The solutions corresponding to other right- hand sides K with K > J may not be guaranteed as well, but only the first such right-hand side is reported. If a small componentwise error is not requested (PARAMS(3) = 0.0) then the Jth right-hand side is the first with a normwise error bound that is not guaranteed (the smallest J such that ERR_BNDS_NORM(J,1) = 0.0). By default (PARAMS(3) = 1.0) the Jth right-hand side is the first with either a normwise or componentwise error bound that is not guaranteed (the smallest J such that either ERR_BNDS_NORM(J,1) = 0.0 or ERR_BNDS_COMP(J,1) = 0.0). See the definition of ERR_BNDS_NORM(:,1) and ERR_BNDS_COMP(:,1). To get information about all of the right-hand sides check ERR_BNDS_NORM or ERR_BNDS_COMP.

Related Information

For this routine in other precisions, please see *cgerfsx*, *dgerfsx* and *zgerfsx*. It also exists with a native C interface as *LAPACKE_sgerfsx*.

4.9.40 sgtrfs

sgtrfs improves the computed solution to a system of linear equations when the coefficient matrix is tridiagonal, and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgtrfs(TRANS, N, NRHS, DL, D, DU, DLF, DF, DUF, DU2, IPIV, B, LDB,
                  X, LDX, FERR, BERR, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgtrfs_(const char *trans, const armpl_int_t *n, const armpl_int_t *nrhs,
             const float *dl, const float *d, const float *du,
             const float *dlf, const float *df, const float *duf,
             const float *du2, const armpl_int_t *ipiv, const float *b,
             const armpl_int_t *ldb, float *x, const armpl_int_t *ldx,
             float *ferr, float *berr, float *work, armpl_int_t *iwork,
             armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose = Transpose)

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

DL Input parameter.

DL is REAL

DL is an array, dimension (N-1). The (n-1) subdiagonal elements of A.

D Input parameter.

D is REAL

D is an array, dimension (N). The diagonal elements of A.

DU Input parameter.

DU is REAL

DU is an array, dimension (N-1). The (n-1) superdiagonal elements of A.

DLF Input parameter.

DLF is REAL

DLF is an array, dimension (N-1). The (n-1) multipliers that define the matrix L from the LU factorization of A as computed by SGTTRF.

DF Input parameter.

DF is REAL

DF is an array, dimension (N). The n diagonal elements of the upper triangular matrix U from the LU factorization of A.

DUF Input parameter.

DUF is REAL

DUF is an array, dimension (N-1). The (n-1) elements of the first superdiagonal of U.

DU2 Input parameter.

DU2 is REAL

DU2 is an array, dimension (N-2). The (n-2) elements of the second superdiagonal of U.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices; for 1 <= i <= n, row i of the matrix was interchanged with row IPIV(i). IPIV(i) will always be either i or i+1; IPIV(i) = i indicates a row interchange was not required.

B Input parameter.

B is REAL

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is REAL

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by SGTTRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgtrfs*, *dgtrfs* and *zgtrfs*. It also exists with a native C interface as *LAPACKE_sgtrfs*.

4.9.41 spbrfs

spbrfs improves the computed solution to a system of linear equations when the coefficient matrix is symmetric positive definite and banded, and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine spbrfs(UPLO, N, KD, NRHS, AB, LDAB, AFB, LDAFB, B, LDB, X, LDX,
                  FERR, BERR, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void spbrfs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
             const armpl_int_t *nrhs, const float *ab,
             const armpl_int_t *ldab, const float *afb,
             const armpl_int_t *ldafb, const float *b, const armpl_int_t *ldb,
             float *x, const armpl_int_t *ldx, float *ferr, float *berr,
             float *work, armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AB Input parameter.

AB is REAL

AB is an array, dimension (LDAB, N). The upper or lower triangle of the symmetric band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

AFB Input parameter.

AFB is REAL

AFB is an array, dimension (LDAFB, N). The triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T of the band matrix A as computed by SPBTRF, in the same storage format as A (see AB).

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AFB. LDAFB >= KD+1.

B Input parameter.

B is REAL

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is REAL

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by SPBTRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cpbrfs*, *dpbrfs* and *zpbrfs*. It also exists with a native C interface as *LAPACKE_spbrfs*.

4.9.42 sporfs

sporfs improves the computed solution to a system of linear equations when the coefficient matrix is symmetric positive definite, and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine sporfs(UPLO, N, NRHS, A, LDA, AF, LDAF, B, LDB, X, LDX, FERR, BERR,
                  WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sporfs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const float *a, const armpl_int_t *lda, const float *af,
             const armpl_int_t *ldaf, const float *b, const armpl_int_t *ldb,
             float *x, const armpl_int_t *ldx, float *ferr, float *berr,
             float *work, armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). The symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is REAL

AF is an array, dimension (LDAF, N). The triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , as computed by SPOTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

B Input parameter.

B is REAL

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is REAL

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by SPOTRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cporfs*, *dporfs* and *zporfs*. It also exists with a native C interface as *LAPACKE_sporfs*.

4.9.43 sporfsx

```
SPORFSX improves the computed solution to a system of linear
   equations when the coefficient matrix is symmetric positive
   definite, and provides error bounds and backward error estimates
   for the solution.  In addition to normwise error bound, the code
   provides maximum componentwise error bound if possible.  See
   comments for ERR_BNDS_NORM and ERR_BNDS_COMP for details of the
   error bounds.

   The original system of linear equations may have been equilibrated
   before calling this routine, as described by arguments EQUED and S
   below. In this case, the solution and error bounds returned are
   for the original unequilibrated system.                          Some optional␣
↪parameters are bundled in the PARAMS array.  These
    settings determine how refinement is performed, but often the
    defaults are acceptable.  If the defaults are acceptable, users
    can pass NPARAMS = 0 which prevents the source code from accessing
    the PARAMS argument.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine sporfsx(UPLO, EQUED, N, NRHS, A, LDA, AF, LDAF, S, B, LDB, X, LDX,
                   RCOND, BERR, N_ERR_BNDS, ERR_BNDS_NORM, ERR_BNDS_COMP,
                   NPARAMS, PARAMS, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sporfsx_(const char *uplo, const char *equed, const armpl_int_t *n,
              const armpl_int_t *nrhs, const float *a, const armpl_int_t *lda,
              const float *af, const armpl_int_t *ldaf, const float *s,
```

(continues on next page)

(continued from previous page)

```
                const float *b, const armpl_int_t *ldb, float *x,
                const armpl_int_t *ldx, float *rcond, float *berr,
                const armpl_int_t *n_err_bnds, float *err_bnds_norm,
                float *err_bnds_comp, const armpl_int_t *nparams, float *params,
                float *work, armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

EQUED Input parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done to A before calling this routine. This is needed to compute the solution and error bounds correctly. = 'N': No equilibration = 'Y': Both row and column equilibration, i.e., A has been replaced by diag(S) * A * diag(S). The right hand side B has been changed accordingly.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). The symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is REAL

AF is an array, dimension (LDAF, N). The triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , as computed by SPOTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

S Input and output parameter.

S is REAL

S is an array, dimension (N). The row scale factors for A. If EQUED = 'Y', A is multiplied on the left and right by diag(S). S is an input argument if FACT = 'F'; otherwise, S is an output argument. If FACT = 'F' and EQUED = 'Y', each element of S must be positive. If S is output, each element of S is a power of the

radix. If S is input, each element of S should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input parameter.

B is REAL

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is REAL

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by SGETRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). Componentwise relative backward error. This is the componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

N_ERR_BNDS Input parameter.

N_ERR_BNDS is INTEGER

Number of error bounds to return for each right hand side and each type (normwise or componentwise). See ERR_BNDS_NORM and ERR_BNDS_COMP below.

ERR_BNDS_NORM Output parameter.

ERR_BNDS_NORM is REAL

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 “Trust/don’t trust” boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch(‘Epsilon’).

err = 2 “Guaranteed” error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch(‘Epsilon’). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch(‘Epsilon’) to determine if the error estimate is “guaranteed”. These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Output parameter.

ERR_BNDS_COMP is REAL

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:, N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 “Trust/don’t trust” boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch(‘Epsilon’).

err = 2 “Guaranteed” error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch(‘Epsilon’). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch(‘Epsilon’) to determine if the error estimate is “guaranteed”. These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

NPARAMS Input parameter.

NPARAMS is INTEGER

Specifies the number of parameters set in PARAMS. If .LE. 0, the PARAMS array is never referenced and default values are used.

PARAMS Input and output parameter.

PARAMS is REAL

PARAMS is an array, dimension NPARAMS. Specifies algorithm parameters. If an entry is .LT. 0.0, then that entry will be filled with default value used for that parameter. Only positions up to NPARAMS are accessed; defaults are used for higher-numbered parameters.

PARAMS(LA_LINRX_ITREF_I = 1) : Whether to perform iterative refinement or not. Default: 1.0 = 0.0 : No refinement is performed, and no error bounds are computed. = 1.0 : Use the double-precision

refinement algorithm, possibly with doubled-single computations if the compilation environment does not support DOUBLE PRECISION. (other values are reserved for future use)

PARAMS(LA_LINRX_ITHRESH_I = 2) : Maximum number of residual computations allowed for refinement. Default: 10 Aggressive: Set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in err_bnds_norm and err_bnds_comp may no longer be trustworthy.

PARAMS(LA_LINRX_CWISE_I = 3) : Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Default: 1.0 (attempt componentwise convergence)

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (4*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. The solution to every right-hand side is guaranteed. < 0: If INFO = -i, the i-th argument had an illegal value > 0 and <= N: U(INFO, INFO) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+J: The solution corresponding to the Jth right-hand side is not guaranteed. The solutions corresponding to other right- hand sides K with K > J may not be guaranteed as well, but only the first such right-hand side is reported. If a small componentwise error is not requested (PARAMS(3) = 0.0) then the Jth right-hand side is the first with a normwise error bound that is not guaranteed (the smallest J such that ERR_BNDS_NORM(J,1) = 0.0). By default (PARAMS(3) = 1.0) the Jth right-hand side is the first with either a normwise or componentwise error bound that is not guaranteed (the smallest J such that either ERR_BNDS_NORM(J,1) = 0.0 or ERR_BNDS_COMP(J,1) = 0.0). See the definition of ERR_BNDS_NORM(:,1) and ERR_BNDS_COMP(:,1). To get information about all of the right-hand sides check ERR_BNDS_NORM or ERR_BNDS_COMP.

Related Information

For this routine in other precisions, please see *cporfsx*, *dporfsx* and *zporfsx*. It also exists with a native C interface as *LAPACKE_sporfsx*.

4.9.44 spprfs

spprfs improves the computed solution to a system of linear equations when the coefficient matrix is symmetric positive definite and packed, and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine spprfs(UPLO, N, NRHS, AP, AFP, B, LDB, X, LDX, FERR, BERR, WORK,
                  IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void spprfs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const float *ap, const float *afp, const float *b,
             const armpl_int_t *ldb, float *x, const armpl_int_t *ldx,
             float *ferr, float *berr, float *work, armpl_int_t *iwork,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AP Input parameter.

AP is REAL

AP is an array, dimension (N*(N+1)/2). The upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

AFP Input parameter.

AFP is REAL

AFP is an array, dimension (N*(N+1)/2). The triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , as computed by SPPTRF/CPPTRF, packed columnwise in a linear array in the same format as A (see AP).

B Input parameter.

B is REAL

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is REAL

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by SPPTRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cpprfs*, *dpprfs* and *zpprfs*. It also exists with a native C interface as *LAPACKE_spprfs*.

4.9.45 sptrfs

sptrfs improves the computed solution to a system of linear equations when the coefficient matrix is symmetric positive definite and tridiagonal, and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine sptrfs(N, NRHS, D, E, DF, EF, B, LDB, X, LDX, FERR, BERR, WORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void sptrfs_(const armpl_int_t *n, const armpl_int_t *nrhs, const float *d,
             const float *e, const float *df, const float *ef, const float *b,
             const armpl_int_t *ldb, float *x, const armpl_int_t *ldx,
             float *ferr, float *berr, float *work, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

D Input parameter.

D is REAL

D is an array, dimension (N). The n diagonal elements of the tridiagonal matrix A.

E Input parameter.

E is REAL

E is an array, dimension (N-1). The (n-1) subdiagonal elements of the tridiagonal matrix A.

DF Input parameter.

DF is REAL

DF is an array, dimension (N). The n diagonal elements of the diagonal matrix D from the factorization computed by SPTTRF.

EF Input parameter.

EF is REAL

EF is an array, dimension (N-1). The (n-1) subdiagonal elements of the unit bidiagonal factor L from the factorization computed by SPTTRF.

B Input parameter.

B is REAL

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is REAL

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by SPTTRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j).

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cptrfs*, *dptrfs* and *zptrfs*. It also exists with a native C interface as *LAPACKE_sptrfs*.

4.9.46 ssprfs

ssprfs improves the computed solution to a system of linear equations when the coefficient matrix is symmetric indefinite and packed, and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssprfs(UPLO, N, NRHS, AP, AFP, IPIV, B, LDB, X, LDX, FERR, BERR,
                  WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ssprfs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const float *ap, const float *afp, const armpl_int_t *ipiv,
             const float *b, const armpl_int_t *ldb, float *x,
             const armpl_int_t *ldx, float *ferr, float *berr, float *work,
             armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AP Input parameter.

AP is REAL

AP is an array, dimension (N*(N+1)/2). The upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n.

AFP Input parameter.

AFP is REAL

AFP is an array, dimension (N*(N+1)/2). The factored form of the matrix A. AFP contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by SSPTRF, stored as a packed triangular matrix.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by SSPTRF.

B Input parameter.

B is REAL

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is REAL

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by SSPTRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csprfs*, *dsprfs* and *zsprfs*. It also exists with a native C interface as *LAPACKE_ssprfs*.

4.9.47 ssyrfs

ssyrfs improves the computed solution to a system of linear equations when the coefficient matrix is symmetric indefinite, and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssyrfs(UPLO, N, NRHS, A, LDA, AF, LDAF, IPIV, B, LDB, X, LDX, FERR,
                  BERR, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ssyrfs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const float *a, const armpl_int_t *lda, const float *af,
             const armpl_int_t *ldaf, const armpl_int_t *ipiv, const float *b,
             const armpl_int_t *ldb, float *x, const armpl_int_t *ldx,
             float *ferr, float *berr, float *work, armpl_int_t *iwork,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). The symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is REAL

AF is an array, dimension (LDAF, N). The factored form of the matrix A. AF contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by SSYTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by SSYTRF.

B Input parameter.

B is REAL

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is REAL

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by SSYTRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csyrfs*, *dsyrfs* and *zsyrfs*. It also exists with a native C interface as *LAPACKE_ssyrfs*.

4.9.48 ssyrfsx

```
SSYRFSX improves the computed solution to a system of linear
   equations when the coefficient matrix is symmetric indefinite, and
   provides error bounds and backward error estimates for the
   solution.  In addition to normwise error bound, the code provides
   maximum componentwise error bound if possible.  See comments for
   ERR_BNDS_NORM and ERR_BNDS_COMP for details of the error bounds.

   The original system of linear equations may have been equilibrated
   before calling this routine, as described by arguments EQUED and S
   below. In this case, the solution and error bounds returned are
   for the original unequilibrated system.                          Some optional␣
↪parameters are bundled in the PARAMS array.  These
    settings determine how refinement is performed, but often the
    defaults are acceptable.  If the defaults are acceptable, users
    can pass NPARAMS = 0 which prevents the source code from accessing
    the PARAMS argument.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine ssyrfsx(UPLO, EQUED, N, NRHS, A, LDA, AF, LDAF, IPIV, S, B, LDB, X,
                   LDX, RCOND, BERR, N_ERR_BNDS, ERR_BNDS_NORM, ERR_BNDS_COMP,
                   NPARAMS, PARAMS, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ssyrfsx_(const char *uplo, const char *equed, const armpl_int_t *n,
              const armpl_int_t *nrhs, const float *a, const armpl_int_t *lda,
              const float *af, const armpl_int_t *ldaf,
              const armpl_int_t *ipiv, const float *s, const float *b,
```

(continues on next page)

(continued from previous page)

```
                const armpl_int_t *ldb, float *x, const armpl_int_t *ldx,
                float *rcond, float *berr, const armpl_int_t *n_err_bnds,
                float *err_bnds_norm, float *err_bnds_comp,
                const armpl_int_t *nparams, float *params, float *work,
                armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

EQUED Input parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done to A before calling this routine. This is needed to compute the solution and error bounds correctly. = 'N': No equilibration = 'Y': Both row and column equilibration, i.e., A has been replaced by diag(S) * A * diag(S). The right hand side B has been changed accordingly.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). The symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is REAL

AF is an array, dimension (LDAF, N). The factored form of the matrix A. AF contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by SSYTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by SSYTRF.

S Input and output parameter.

S is REAL

S is an array, dimension (N). The scale factors for A. If EQUED = 'Y', A is multiplied on the left and right by diag(S). S is an input argument if FACT = 'F'; otherwise, S is an output argument. If FACT = 'F' and EQUED = 'Y', each element of S must be positive. If S is output, each element of S is a power of the radix. If S is input, each element of S should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input parameter.

B is REAL

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is REAL

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by SGETRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is REAL

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). Componentwise relative backward error. This is the componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

N_ERR_BNDS Input parameter.

N_ERR_BNDS is INTEGER

Number of error bounds to return for each right hand side and each type (normwise or componentwise). See ERR_BNDS_NORM and ERR_BNDS_COMP below.

ERR_BNDS_NORM Output parameter.

ERR_BNDS_NORM is REAL

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Output parameter.

ERR_BNDS_COMP is REAL

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:, N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

NPARAMS Input parameter.

NPARAMS is INTEGER

Specifies the number of parameters set in PARAMS. If .LE. 0, the PARAMS array is never referenced and default values are used.

PARAMS Input and output parameter.

PARAMS is REAL

PARAMS is an array, dimension NPARAMS. Specifies algorithm parameters. If an entry is .LT. 0.0, then that entry will be filled with default value used for that parameter. Only positions up to NPARAMS are accessed; defaults are used for higher-numbered parameters.

PARAMS(LA_LINRX_ITREF_I = 1) : Whether to perform iterative refinement or not. Default: 1.0 = 0.0 : No refinement is performed, and no error bounds are computed. = 1.0 : Use the double-precision refinement algorithm, possibly with doubled-single computations if the compilation environment does not support DOUBLE PRECISION. (other values are reserved for future use)

PARAMS(LA_LINRX_ITHRESH_I = 2) : Maximum number of residual computations allowed for refinement. Default: 10 Aggressive: Set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in err_bnds_norm and err_bnds_comp may no longer be trustworthy.

PARAMS(LA_LINRX_CWISE_I = 3) : Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Default: 1.0 (attempt componentwise convergence)

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (4*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. The solution to every right-hand side is guaranteed. < 0: If INFO = -i, the i-th argument had an illegal value > 0 and <= N: U(INFO, INFO) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+J: The solution corresponding to the Jth right-hand side is not guaranteed. The solutions corresponding to other right- hand sides K with K > J may not be guaranteed as well, but only the first such right-hand side is reported. If a small componentwise error is not requested (PARAMS(3) = 0.0) then the Jth right-hand side is the first with a normwise error bound that is not guaranteed (the smallest J such that ERR_BNDS_NORM(J,1) = 0.0). By default (PARAMS(3) = 1.0) the Jth right-hand side is the first with either a normwise or componentwise error bound that is not guaranteed (the smallest J such that either ERR_BNDS_NORM(J,1) = 0.0 or ERR_BNDS_COMP(J,1) = 0.0). See the definition of ERR_BNDS_NORM(:,1) and ERR_BNDS_COMP(:,1). To get information about all of the right-hand sides check ERR_BNDS_NORM or ERR_BNDS_COMP.

Related Information

For this routine in other precisions, please see *csyrfsx*, *dsyrfsx* and *zsyrfsx*. It also exists with a native C interface as *LAPACKE_ssyrfsx*.

4.9.49 stbrfs

stbrfs provides error bounds and backward error estimates for the solution to a system of linear equations with a triangular band coefficient matrix.

The solution matrix X must be computed by STBTRS or some other means before entering this routine. stbrfs does not do iterative refinement because doing so cannot improve the backward error.

Syntax

Fortran specification:

```
use armpl_library

subroutine stbrfs(UPLO, TRANS, DIAG, N, KD, NRHS, AB, LDAB, B, LDB, X, LDX,
                  FERR, BERR, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void stbrfs_(const char *uplo, const char *trans, const char *diag,
             const armpl_int_t *n, const armpl_int_t *kd,
             const armpl_int_t *nrhs, const float *ab,
             const armpl_int_t *ldab, const float *b, const armpl_int_t *ldb,
             const float *x, const armpl_int_t *ldx, float *ferr, float *berr,
             float *work, armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose = Transpose)

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals or subdiagonals of the triangular band matrix A. KD >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AB Input parameter.

AB is REAL

AB is an array, dimension (LDAB, N). The upper or lower triangular band matrix A, stored in the first kd+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd). If DIAG = 'U', the diagonal elements of A are not referenced and are assumed to be 1.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

B Input parameter.

B is REAL

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input parameter.

X is REAL

X is an array, dimension (LDX, NRHS). The solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctbrfs*, *dtbrfs* and *ztbrfs*. It also exists with a native C interface as *LAPACKE_stbrfs*.

4.9.50 stprfs

stprfs provides error bounds and backward error estimates for the solution to a system of linear equations with a triangular packed coefficient matrix.

The solution matrix X must be computed by STPTRS or some other means before entering this routine. stprfs does not do iterative refinement because doing so cannot improve the backward error.

Syntax

Fortran specification:

```
use armpl_library

subroutine stprfs(UPLO, TRANS, DIAG, N, NRHS, AP, B, LDB, X, LDX, FERR, BERR,
                  WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void stprfs_(const char *uplo, const char *trans, const char *diag,
             const armpl_int_t *n, const armpl_int_t *nrhs, const float *ap,
             const float *b, const armpl_int_t *ldb, const float *x,
             const armpl_int_t *ldx, float *ferr, float *berr, float *work,
             armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose = Transpose)

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AP Input parameter.

AP is REAL

AP is an array, dimension (N*(N+1)/2). The upper or lower triangular matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n. If DIAG = 'U', the diagonal elements of A are not referenced and are assumed to be 1.

B Input parameter.

B is REAL

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input parameter.

X is REAL

X is an array, dimension (LDX, NRHS). The solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctprfs*, *dtprfs* and *ztprfs*. It also exists with a native C interface as *LAPACKE_stprfs*.

4.9.51 strrfs

strrfs provides error bounds and backward error estimates for the solution to a system of linear equations with a triangular coefficient matrix.

The solution matrix X must be computed by STRTRS or some other means before entering this routine. strrfs does not do iterative refinement because doing so cannot improve the backward error.

Syntax

Fortran specification:

```
use armpl_library

subroutine strrfs(UPLO, TRANS, DIAG, N, NRHS, A, LDA, B, LDB, X, LDX, FERR,
                  BERR, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void strrfs_(const char *uplo, const char *trans, const char *diag,
             const armpl_int_t *n, const armpl_int_t *nrhs, const float *a,
             const armpl_int_t *lda, const float *b, const armpl_int_t *ldb,
             const float *x, const armpl_int_t *ldx, float *ferr, float *berr,
             float *work, armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose = Transpose)

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). The triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input parameter.

B is REAL

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input parameter.

X is REAL

X is an array, dimension (LDX, NRHS). The solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is REAL

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctrrfs*, *dtrrfs* and *ztrrfs*. It also exists with a native C interface as *LAPACKE_strrfs*.

4.9.52 zgbrfs

zgbrfs improves the computed solution to a system of linear equations when the coefficient matrix is banded, and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgbrfs(TRANS, N, KL, KU, NRHS, AB, LDAB, AFB, LDAFB, IPIV, B, LDB,
                  X, LDX, FERR, BERR, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgbrfs_(const char *trans, const armpl_int_t *n, const armpl_int_t *kl,
             const armpl_int_t *ku, const armpl_int_t *nrhs,
             const armpl_doublecomplex_t *ab, const armpl_int_t *ldab,
             const armpl_doublecomplex_t *afb, const armpl_int_t *ldafb,
             const armpl_int_t *ipiv, const armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, armpl_doublecomplex_t *x,
             const armpl_int_t *ldx, double *ferr, double *berr,
             armpl_doublecomplex_t *work, double *rwork, armpl_int_t *info,
             ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose)

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AB Input parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). The original band matrix A, stored in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(ku+1+i-j,j) = A(i,j) for max(1,j-ku)<=i<=min(n,j+kl).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KL+KU+1.

AFB Input parameter.

AFB is COMPLEX*16

AFB is an array, dimension (LDAFB, N). Details of the LU factorization of the band matrix A, as computed by ZGBTRF. U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1.

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AFB. LDAFB >= 2*KL*KU+1.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from ZGBTRF; for 1<=i<=N, row i of the matrix was interchanged with row IPIV(i).

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by ZGBTRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgbrfs*, *dgbrfs* and *sgbrfs*. It also exists with a native C interface as *LAPACKE_zgbrfs*.

4.9.53 zgbrfsx

```
ZGBRFSX improves the computed solution to a system of linear
   equations and provides error bounds and backward error estimates
   for the solution.  In addition to normwise error bound, the code
   provides maximum componentwise error bound if possible.  See
   comments for ERR_BNDS_NORM and ERR_BNDS_COMP for details of the
   error bounds.

   The original system of linear equations may have been equilibrated
   before calling this routine, as described by arguments EQUED, R
   and C below. In this case, the solution and error bounds returned
   are for the original unequilibrated system.                                Some␣
↪optional parameters are bundled in the PARAMS array.  These
    settings determine how refinement is performed, but often the
    defaults are acceptable.  If the defaults are acceptable, users
    can pass NPARAMS = 0 which prevents the source code from accessing
    the PARAMS argument.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zgbrfsx(TRANS, EQUED, N, KL, KU, NRHS, AB, LDAB, AFB, LDAFB, IPIV,
                   R, C, B, LDB, X, LDX, RCOND, BERR, N_ERR_BNDS,
                   ERR_BNDS_NORM, ERR_BNDS_COMP, NPARAMS, PARAMS, WORK, RWORK,
                   INFO)
```

C specification:

```
#include "armpl.h"

void zgbrfsx_(const char *trans, const char *equed, const armpl_int_t *n,
              const armpl_int_t *kl, const armpl_int_t *ku,
              const armpl_int_t *nrhs, const armpl_doublecomplex_t *ab,
              const armpl_int_t *ldab, const armpl_doublecomplex_t *afb,
              const armpl_int_t *ldafb, const armpl_int_t *ipiv,
              const double *r, const double *c,
              const armpl_doublecomplex_t *b, const armpl_int_t *ldb,
              armpl_doublecomplex_t *x, const armpl_int_t *ldx, double *rcond,
              double *berr, const armpl_int_t *n_err_bnds,
              double *err_bnds_norm, double *err_bnds_comp,
              const armpl_int_t *nparams, double *params,
              armpl_doublecomplex_t *work, double *rwork, armpl_int_t *info,
              ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose = Transpose)

EQUED Input parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done to A before calling this routine. This is needed to compute the solution and error bounds correctly. = 'N': No equilibration = 'R': Row equilibration, i.e., A has been premultiplied by diag(R). = 'C': Column equilibration, i.e., A has been postmultiplied by diag(C). = 'B': Both row and column equilibration, i.e., A has been replaced by diag(R) * A * diag(C). The right hand side B has been changed accordingly.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AB Input parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). The original band matrix A, stored in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(ku+1+i-j,j) = A(i,j) for max(1,j-ku)<=i<=min(n,j+kl).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KL+KU+1.

AFB Input parameter.

AFB is COMPLEX*16

AFB is an array, dimension (LDAFB, N). Details of the LU factorization of the band matrix A, as computed by DGBTRF. U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1.

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AFB. LDAFB >= 2*KL*KU+1.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from DGETRF; for 1<=i<=N, row i of the matrix was interchanged with row IPIV(i).

R Input and output parameter.

R is DOUBLE PRECISION

R is an array, dimension (N). The row scale factors for A. If EQUED = 'R' or 'B', A is multiplied on the left by diag(R); if EQUED = 'N' or 'C', R is not accessed. R is an input argument if FACT = 'F'; otherwise, R is an output argument. If FACT = 'F' and EQUED = 'R' or 'B', each element of R must be positive. If R is output, each element of R is a power of the radix. If R is input, each element of R should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). The column scale factors for A. If EQUED = 'C' or 'B', A is multiplied on the right by diag(C); if EQUED = 'N' or 'R', C is not accessed. C is an input argument if FACT = 'F'; otherwise, C is an output argument. If FACT = 'F' and EQUED = 'C' or 'B', each element of C must be positive. If C is output, each element of C is a power of the radix. If C is input, each element of C should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by DGETRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). Componentwise relative backward error. This is the componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

N_ERR_BNDS Input parameter.

N_ERR_BNDS is INTEGER

Number of error bounds to return for each right hand side and each type (normwise or componentwise). See ERR_BNDS_NORM and ERR_BNDS_COMP below.

ERR_BNDS_NORM Output parameter.

ERR_BNDS_NORM is DOUBLE PRECISION

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * dlamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * dlamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * dlamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Output parameter.

ERR_BNDS_COMP is DOUBLE PRECISION

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:, N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * dlamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * dlamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * dlamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix

Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

NPARAMS Input parameter.

NPARAMS is INTEGER

Specifies the number of parameters set in PARAMS. If .LE. 0, the PARAMS array is never referenced and default values are used.

PARAMS Input and output parameter.

PARAMS is DOUBLE PRECISION

PARAMS is an array, dimension NPARAMS. Specifies algorithm parameters. If an entry is .LT. 0.0, then that entry will be filled with default value used for that parameter. Only positions up to NPARAMS are accessed; defaults are used for higher-numbered parameters.

PARAMS(LA_LINRX_ITREF_I = 1) : Whether to perform iterative refinement or not. Default: 1.0D+0 = 0.0 : No refinement is performed, and no error bounds are computed. = 1.0 : Use the double-precision refinement algorithm, possibly with doubled-single computations if the compilation environment does not support DOUBLE PRECISION. (other values are reserved for future use)

PARAMS(LA_LINRX_ITHRESH_I = 2) : Maximum number of residual computations allowed for refinement. Default: 10 Aggressive: Set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in err_bnds_norm and err_bnds_comp may no longer be trustworthy.

PARAMS(LA_LINRX_CWISE_I = 3) : Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Default: 1.0 (attempt componentwise convergence)

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. The solution to every right-hand side is guaranteed. < 0: If INFO = -i, the i-th argument had an illegal value > 0 and <= N: U(INFO, INFO) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+J: The solution corresponding to the Jth right-hand side is not guaranteed. The solutions corresponding to other right- hand sides K with K > J may not be guaranteed as well, but only the first such right-hand side is reported. If a small componentwise error is not requested (PARAMS(3) = 0.0) then the Jth right-hand side is the first with a normwise error bound that is not guaranteed (the smallest J such that ERR_BNDS_NORM(J,1) = 0.0). By default (PARAMS(3) = 1.0) the Jth right-hand side is the first with either a normwise or componentwise error bound that is not guaranteed (the smallest J such that either ERR_BNDS_NORM(J,1) = 0.0 or ERR_BNDS_COMP(J,1) = 0.0). See the definition of ERR_BNDS_NORM(:,1) and ERR_BNDS_COMP(:,1). To get information about all of the right-hand sides check ERR_BNDS_NORM or ERR_BNDS_COMP.

Related Information

For this routine in other precisions, please see *cgbrfsx*, *dgbrfsx* and *sgbrfsx*. It also exists with a native C interface as *LAPACKE_zgbrfsx*.

4.9.54 zgerfs

`zgerfs` improves the computed solution to a system of linear equations and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgerfs(TRANS, N, NRHS, A, LDA, AF, LDAF, IPIV, B, LDB, X, LDX,
                  FERR, BERR, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgerfs_(const char *trans, const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_doublecomplex_t *af, const armpl_int_t *ldaf,
             const armpl_int_t *ipiv, const armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, armpl_doublecomplex_t *x,
             const armpl_int_t *ldx, double *ferr, double *berr,
             armpl_doublecomplex_t *work, double *rwork, armpl_int_t *info,
             ... );
```

Parameters

TRANS Input parameter.

`TRANS` is `CHARACTER*1`

Specifies the form of the system of equations: = 'N': `A` * `X` = `B` (No transpose) = 'T': A^T * `X` = `B` (Transpose) = 'C': A^H * `X` = `B` (Conjugate transpose)

N Input parameter.

`N` is `INTEGER`

The order of the matrix `A`. `N` >= 0.

NRHS Input parameter.

`NRHS` is `INTEGER`

The number of right hand sides, i.e., the number of columns of the matrices `B` and `X`. `NRHS` >= 0.

A Input parameter.

`A` is `COMPLEX*16`

`A` is an array, dimension (`LDA`, `N`). The original N-by-N matrix `A`.

LDA Input parameter.

`LDA` is `INTEGER`

The leading dimension of the array `A`. `LDA` >= max(1, `N`).

AF Input parameter.

`AF` is `COMPLEX*16`

`AF` is an array, dimension (`LDAF`, `N`). The factors L and U from the factorization `A` = P*L*U as computed by ZGETRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from ZGETRF; for 1<=i<=N, row i of the matrix was interchanged with row IPIV(i).

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by ZGETRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgerfs*, *dgerfs* and *sgerfs*. It also exists with a native C interface as *LAPACKE_zgerfs*.

4.9.55 zgerfsx

```
ZGERFSX improves the computed solution to a system of linear
   equations and provides error bounds and backward error estimates
   for the solution.  In addition to normwise error bound, the code
   provides maximum componentwise error bound if possible.  See
   comments for ERR_BNDS_NORM and ERR_BNDS_COMP for details of the
   error bounds.

   The original system of linear equations may have been equilibrated
   before calling this routine, as described by arguments EQUED, R
   and C below. In this case, the solution and error bounds returned
   are for the original unequilibrated system.                    Some␣
↪optional parameters are bundled in the PARAMS array.  These
    settings determine how refinement is performed, but often the
    defaults are acceptable.  If the defaults are acceptable, users
    can pass NPARAMS = 0 which prevents the source code from accessing
    the PARAMS argument.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zgerfsx(TRANS, EQUED, N, NRHS, A, LDA, AF, LDAF, IPIV, R, C, B,
                   LDB, X, LDX, RCOND, BERR, N_ERR_BNDS, ERR_BNDS_NORM,
                   ERR_BNDS_COMP, NPARAMS, PARAMS, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgerfsx_(const char *trans, const char *equed, const armpl_int_t *n,
              const armpl_int_t *nrhs, const armpl_doublecomplex_t *a,
              const armpl_int_t *lda, const armpl_doublecomplex_t *af,
              const armpl_int_t *ldaf, const armpl_int_t *ipiv,
              const double *r, const double *c,
              const armpl_doublecomplex_t *b, const armpl_int_t *ldb,
              armpl_doublecomplex_t *x, const armpl_int_t *ldx, double *rcond,
              double *berr, const armpl_int_t *n_err_bnds,
              double *err_bnds_norm, double *err_bnds_comp,
              const armpl_int_t *nparams, double *params,
              armpl_doublecomplex_t *work, double *rwork, armpl_int_t *info,
              ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose = Transpose)

EQUED Input parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done to A before calling this routine. This is needed to compute the solution and error bounds correctly. = 'N': No equilibration = 'R': Row equilibration, i.e., A has been premultiplied by diag(R). = 'C': Column equilibration, i.e., A has been postmultiplied by diag(C). = 'B': Both row and column equilibration, i.e., A has been replaced by diag(R) * A * diag(C). The right hand side B has been changed accordingly.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The original N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX*16

AF is an array, dimension (LDAF, N). The factors L and U from the factorization A = P*L*U as computed by ZGETRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from ZGETRF; for 1<=i<=N, row i of the matrix was interchanged with row IPIV(i).

R Input parameter.

R is DOUBLE PRECISION

R is an array, dimension (N). The row scale factors for A. If EQUED = 'R' or 'B', A is multiplied on the left by diag(R); if EQUED = 'N' or 'C', R is not accessed. If R is accessed, each element of R should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

C Input parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). The column scale factors for A. If EQUED = 'C' or 'B', A is multiplied on the right by diag(C); if EQUED = 'N' or 'R', C is not accessed. If C is accessed, each element of C should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by ZGETRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). Componentwise relative backward error. This is the componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

N_ERR_BNDS Input parameter.

N_ERR_BNDS is INTEGER

Number of error bounds to return for each right hand side and each type (normwise or componentwise). See ERR_BNDS_NORM and ERR_BNDS_COMP below.

ERR_BNDS_NORM Output parameter.

ERR_BNDS_NORM is DOUBLE PRECISION

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 “Trust/don’t trust” boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * dlamch(‘Epsilon’).

err = 2 “Guaranteed” error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * dlamch(‘Epsilon’). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * dlamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Output parameter.

ERR_BNDS_COMP is DOUBLE PRECISION

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:, N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * dlamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * dlamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * dlamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

NPARAMS Input parameter.

NPARAMS is INTEGER

Specifies the number of parameters set in PARAMS. If .LE. 0, the PARAMS array is never referenced and default values are used.

PARAMS Input and output parameter.

PARAMS is DOUBLE PRECISION

PARAMS is an array, dimension NPARAMS. Specifies algorithm parameters. If an entry is .LT. 0.0, then that entry will be filled with default value used for that parameter. Only positions up to NPARAMS are accessed; defaults are used for higher-numbered parameters.

PARAMS(LA_LINRX_ITREF_I = 1) : Whether to perform iterative refinement or not. Default: 1.0D+0 = 0.0 : No refinement is performed, and no error bounds are computed. = 1.0 : Use the double-precision refinement algorithm, possibly with doubled-single computations if the compilation environment does not support DOUBLE PRECISION. (other values are reserved for future use)

PARAMS(LA_LINRX_ITHRESH_I = 2) : Maximum number of residual computations allowed for refinement. Default: 10 Aggressive: Set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in err_bnds_norm and err_bnds_comp may no longer be trustworthy.

PARAMS(LA_LINRX_CWISE_I = 3) : Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Default: 1.0 (attempt componentwise convergence)

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. The solution to every right-hand side is guaranteed. < 0: If INFO = -i, the i-th argument had an illegal value > 0 and <= N: U(INFO, INFO) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+J: The solution corresponding to the Jth right-hand side is not guaranteed. The solutions corresponding to other right- hand sides K with K > J may not be guaranteed as well, but only the first such right-hand side is reported. If a small componentwise error is not requested (PARAMS(3) = 0.0) then the Jth right-hand side is the first with a normwise error bound that is not guaranteed (the smallest J such that ERR_BNDS_NORM(J,1) = 0.0). By default (PARAMS(3) = 1.0) the Jth right-hand side is the first with either a normwise or componentwise error bound that is not guaranteed (the smallest J such that either ERR_BNDS_NORM(J,1) = 0.0 or ERR_BNDS_COMP(J,1) = 0.0). See the definition of ERR_BNDS_NORM(:,1) and ERR_BNDS_COMP(:,1). To get information about all of the right-hand sides check ERR_BNDS_NORM or ERR_BNDS_COMP.

Related Information

For this routine in other precisions, please see *cgerfsx*, *dgerfsx* and *sgerfsx*. It also exists with a native C interface as *LAPACKE_zgerfsx*.

4.9.56 zgtrfs

zgtrfs improves the computed solution to a system of linear equations when the coefficient matrix is tridiagonal, and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgtrfs(TRANS, N, NRHS, DL, D, DU, DLF, DF, DUF, DU2, IPIV, B, LDB,
                  X, LDX, FERR, BERR, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgtrfs_(const char *trans, const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_doublecomplex_t *dl, const armpl_doublecomplex_t *d,
             const armpl_doublecomplex_t *du,
             const armpl_doublecomplex_t *dlf,
             const armpl_doublecomplex_t *df,
```

(continues on next page)

(continued from previous page)

```
                const armpl_doublecomplex_t *duf,
                const armpl_doublecomplex_t *du2, const armpl_int_t *ipiv,
                const armpl_doublecomplex_t *b, const armpl_int_t *ldb,
                armpl_doublecomplex_t *x, const armpl_int_t *ldx, double *ferr,
                double *berr, armpl_doublecomplex_t *work, double *rwork,
                armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose)

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

DL Input parameter.

DL is COMPLEX*16

DL is an array, dimension (N-1). The (n-1) subdiagonal elements of A.

D Input parameter.

D is COMPLEX*16

D is an array, dimension (N). The diagonal elements of A.

DU Input parameter.

DU is COMPLEX*16

DU is an array, dimension (N-1). The (n-1) superdiagonal elements of A.

DLF Input parameter.

DLF is COMPLEX*16

DLF is an array, dimension (N-1). The (n-1) multipliers that define the matrix L from the LU factorization of A as computed by ZGTTRF.

DF Input parameter.

DF is COMPLEX*16

DF is an array, dimension (N). The n diagonal elements of the upper triangular matrix U from the LU factorization of A.

DUF Input parameter.

DUF is COMPLEX*16

DUF is an array, dimension (N-1). The (n-1) elements of the first superdiagonal of U.

DU2 Input parameter.

DU2 is COMPLEX*16

DU2 is an array, dimension (N-2). The (n-2) elements of the second superdiagonal of U.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices; for 1 <= i <= n, row i of the matrix was interchanged with row IPIV(i). IPIV(i) will always be either i or i+1; IPIV(i) = i indicates a row interchange was not required.

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by ZGTTRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgtrfs*, *dgtrfs* and *sgtrfs*. It also exists with a native C interface as *LAPACKE_zgtrfs*.

4.9.57 zherfs

zherfs improves the computed solution to a system of linear equations when the coefficient matrix is Hermitian indefinite, and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine zherfs(UPLO, N, NRHS, A, LDA, AF, LDAF, IPIV, B, LDB, X, LDX, FERR,
                  BERR, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zherfs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_doublecomplex_t *af, const armpl_int_t *ldaf,
             const armpl_int_t *ipiv, const armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, armpl_doublecomplex_t *x,
             const armpl_int_t *ldx, double *ferr, double *berr,
             armpl_doublecomplex_t *work, double *rwork, armpl_int_t *info,
             ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX*16

AF is an array, dimension (LDAF, N). The factored form of the matrix A. AF contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^H or A = L*D*L^H as computed by ZHETRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZHETRF.

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by ZHETRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cherfs*. It also exists with a native C interface as *LAPACKE_zherfs*.

4.9.58 zherfsx

```
ZHERFSX improves the computed solution to a system of linear
   equations when the coefficient matrix is Hermitian indefinite, and
   provides error bounds and backward error estimates for the
   solution.  In addition to normwise error bound, the code provides
   maximum componentwise error bound if possible.  See comments for
   ERR_BNDS_NORM and ERR_BNDS_COMP for details of the error bounds.

   The original system of linear equations may have been equilibrated
   before calling this routine, as described by arguments EQUED and S
   below. In this case, the solution and error bounds returned are
   for the original unequilibrated system.                          Some optional␣
↪parameters are bundled in the PARAMS array.  These
    settings determine how refinement is performed, but often the
    defaults are acceptable.  If the defaults are acceptable, users
    can pass NPARAMS = 0 which prevents the source code from accessing
    the PARAMS argument.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zherfsx(UPLO, EQUED, N, NRHS, A, LDA, AF, LDAF, IPIV, S, B, LDB, X,
                   LDX, RCOND, BERR, N_ERR_BNDS, ERR_BNDS_NORM, ERR_BNDS_COMP,
                   NPARAMS, PARAMS, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zherfsx_(const char *uplo, const char *equed, const armpl_int_t *n,
              const armpl_int_t *nrhs, const armpl_doublecomplex_t *a,
              const armpl_int_t *lda, const armpl_doublecomplex_t *af,
              const armpl_int_t *ldaf, const armpl_int_t *ipiv,
              const double *s, const armpl_doublecomplex_t *b,
              const armpl_int_t *ldb, armpl_doublecomplex_t *x,
              const armpl_int_t *ldx, double *rcond, double *berr,
              const armpl_int_t *n_err_bnds, double *err_bnds_norm,
              double *err_bnds_comp, const armpl_int_t *nparams,
              double *params, armpl_doublecomplex_t *work, double *rwork,
              armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

EQUED Input parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done to A before calling this routine. This is needed to compute the solution and error bounds correctly. = 'N': No equilibration = 'Y': Both row and column equilibration, i.e., A has been replaced by diag(S) * A * diag(S). The right hand side B has been changed accordingly.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX*16

AF is an array, dimension (LDAF, N). The factored form of the matrix A. AF contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by DSYTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by DSYTRF.

S Input and output parameter.

S is DOUBLE PRECISION

S is an array, dimension (N). The scale factors for A. If EQUED = 'Y', A is multiplied on the left and right by diag(S). S is an input argument if FACT = 'F'; otherwise, S is an output argument. If FACT = 'F' and EQUED = 'Y', each element of S must be positive. If S is output, each element of S is a power of the radix. If S is input, each element of S should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by DGETRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). Componentwise relative backward error. This is the componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

N_ERR_BNDS Input parameter.

N_ERR_BNDS is INTEGER

Number of error bounds to return for each right hand side and each type (normwise or componentwise). See ERR_BNDS_NORM and ERR_BNDS_COMP below.

ERR_BNDS_NORM Output parameter.

ERR_BNDS_NORM is DOUBLE PRECISION

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 “Trust/don’t trust” boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * dlamch(‘Epsilon’).

err = 2 “Guaranteed” error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * dlamch(‘Epsilon’). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * dlamch(‘Epsilon’) to determine if the error estimate is “guaranteed”. These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Output parameter.

ERR_BNDS_COMP is DOUBLE PRECISION

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:, N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * dlamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * dlamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * dlamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

NPARAMS Input parameter.

NPARAMS is INTEGER

Specifies the number of parameters set in PARAMS. If .LE. 0, the PARAMS array is never referenced and default values are used.

PARAMS Input and output parameter.

PARAMS is DOUBLE PRECISION

PARAMS is an array, dimension NPARAMS. Specifies algorithm parameters. If an entry is .LT. 0.0, then that entry will be filled with default value used for that parameter. Only positions up to NPARAMS are accessed; defaults are used for higher-numbered parameters.

PARAMS(LA_LINRX_ITREF_I = 1) : Whether to perform iterative refinement or not. Default: 1.0D+0 = 0.0 : No refinement is performed, and no error bounds are computed. = 1.0 : Use the double-precision refinement algorithm, possibly with doubled-single computations if the compilation environment does not support DOUBLE PRECISION. (other values are reserved for future use)

PARAMS(LA_LINRX_ITHRESH_I = 2) : Maximum number of residual computations allowed for refinement. Default: 10 Aggressive: Set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in err_bnds_norm and err_bnds_comp may no longer be trustworthy.

PARAMS(LA_LINRX_CWISE_I = 3) : Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Default: 1.0 (attempt componentwise convergence)

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. The solution to every right-hand side is guaranteed. < 0: If INFO = -i, the i-th argument had an illegal value > 0 and <= N: U(INFO, INFO) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+J: The solution corresponding to the Jth right-hand side is not guaranteed. The solutions corresponding to other right- hand sides K with K > J may not be guaranteed as well, but only the first such right-hand side is reported. If a small componentwise error is not requested (PARAMS(3) = 0.0) then the Jth right-hand side is the first with a normwise error bound that is not guaranteed (the smallest J such that ERR_BNDS_NORM(J,1) = 0.0). By default (PARAMS(3) = 1.0) the Jth right-hand side is the first with either a normwise or componentwise error bound that is not guaranteed (the smallest J such that either ERR_BNDS_NORM(J,1) = 0.0 or ERR_BNDS_COMP(J,1) = 0.0). See the definition of ERR_BNDS_NORM(:,1) and ERR_BNDS_COMP(:,1). To get information about all of the right-hand sides check ERR_BNDS_NORM or ERR_BNDS_COMP.

Related Information

For this routine in other precisions, please see *cherfsx*. It also exists with a native C interface as *LAPACKE_zherfsx*.

4.9.59 zhprfs

zhprfs improves the computed solution to a system of linear equations when the coefficient matrix is Hermitian indefinite and packed, and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhprfs(UPLO, N, NRHS, AP, AFP, IPIV, B, LDB, X, LDX, FERR, BERR,
                  WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhprfs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_doublecomplex_t *ap,
             const armpl_doublecomplex_t *afp, const armpl_int_t *ipiv,
             const armpl_doublecomplex_t *b, const armpl_int_t *ldb,
             armpl_doublecomplex_t *x, const armpl_int_t *ldx, double *ferr,
             double *berr, armpl_doublecomplex_t *work, double *rwork,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AP Input parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). The upper or lower triangle of the Hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n.

AFP Input parameter.

AFP is COMPLEX*16

AFP is an array, dimension (N*(N+1)/2). The factored form of the matrix A. AFP contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^H or A = L*D*L^H as computed by ZHPTRF, stored as a packed triangular matrix.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZHPTRF.

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by ZHPTRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *chprfs*. It also exists with a native C interface as *LAPACKE_zhprfs*.

4.9.60 zpbrfs

zpbrfs improves the computed solution to a system of linear equations when the coefficient matrix is Hermitian positive definite and banded, and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine zpbrfs(UPLO, N, KD, NRHS, AB, LDAB, AFB, LDAFB, B, LDB, X, LDX,
                  FERR, BERR, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zpbrfs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
             const armpl_int_t *nrhs, const armpl_doublecomplex_t *ab,
             const armpl_int_t *ldab, const armpl_doublecomplex_t *afb,
             const armpl_int_t *ldafb, const armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, armpl_doublecomplex_t *x,
             const armpl_int_t *ldx, double *ferr, double *berr,
             armpl_doublecomplex_t *work, double *rwork, armpl_int_t *info,
             ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AB Input parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). The upper or lower triangle of the Hermitian band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

AFB Input parameter.

AFB is COMPLEX*16

AFB is an array, dimension (LDAFB, N). The triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H of the band matrix A as computed by ZPBTRF, in the same storage format as A (see AB).

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AFB. LDAFB >= KD+1.

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by ZPBTRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cpbrfs*, *dpbrfs* and *spbrfs*. It also exists with a native C interface as *LAPACKE_zpbrfs*.

4.9.61 zporfs

zporfs improves the computed solution to a system of linear equations when the coefficient matrix is Hermitian positive definite, and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine zporfs(UPLO, N, NRHS, A, LDA, AF, LDAF, B, LDB, X, LDX, FERR, BERR,
                  WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zporfs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_doublecomplex_t *af, const armpl_int_t *ldaf,
             const armpl_doublecomplex_t *b, const armpl_int_t *ldb,
             armpl_doublecomplex_t *x, const armpl_int_t *ldx, double *ferr,
```

(continues on next page)

(continued from previous page)

```
                double *berr, armpl_doublecomplex_t *work, double *rwork,
                armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX*16

AF is an array, dimension (LDAF, N). The triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H , as computed by ZPOTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by ZPOTRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cporfs*, *dporfs* and *sporfs*. It also exists with a native C interface as *LAPACKE_zporfs*.

4.9.62 zporfsx

```
ZPORFSX improves the computed solution to a system of linear
   equations when the coefficient matrix is symmetric positive
   definite, and provides error bounds and backward error estimates
   for the solution.  In addition to normwise error bound, the code
   provides maximum componentwise error bound if possible.  See
   comments for ERR_BNDS_NORM and ERR_BNDS_COMP for details of the
   error bounds.

   The original system of linear equations may have been equilibrated
   before calling this routine, as described by arguments EQUED and S
   below. In this case, the solution and error bounds returned are
   for the original unequilibrated system.                          Some optional␣
↪parameters are bundled in the PARAMS array.  These
    settings determine how refinement is performed, but often the
    defaults are acceptable.  If the defaults are acceptable, users
    can pass NPARAMS = 0 which prevents the source code from accessing
    the PARAMS argument.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zporfsx(UPLO, EQUED, N, NRHS, A, LDA, AF, LDAF, S, B, LDB, X, LDX,
                   RCOND, BERR, N_ERR_BNDS, ERR_BNDS_NORM, ERR_BNDS_COMP,
                   NPARAMS, PARAMS, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zporfsx_(const char *uplo, const char *equed, const armpl_int_t *n,
              const armpl_int_t *nrhs, const armpl_doublecomplex_t *a,
              const armpl_int_t *lda, const armpl_doublecomplex_t *af,
              const armpl_int_t *ldaf, const double *s,
              const armpl_doublecomplex_t *b, const armpl_int_t *ldb,
              armpl_doublecomplex_t *x, const armpl_int_t *ldx, double *rcond,
              double *berr, const armpl_int_t *n_err_bnds,
              double *err_bnds_norm, double *err_bnds_comp,
              const armpl_int_t *nparams, double *params,
              armpl_doublecomplex_t *work, double *rwork, armpl_int_t *info,
              ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

EQUED Input parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done to A before calling this routine. This is needed to compute the solution and error bounds correctly. = 'N': No equilibration = 'Y': Both row and column equilibration, i.e., A has been replaced by diag(S) * A * diag(S). The right hand side B has been changed accordingly.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX*16

AF is an array, dimension (LDAF, N). The triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , as computed by DPOTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

S Input and output parameter.

S is DOUBLE PRECISION

S is an array, dimension (N). The row scale factors for A. If EQUED = 'Y', A is multiplied on the left and right by diag(S). S is an input argument if FACT = 'F'; otherwise, S is an output argument. If FACT = 'F' and EQUED = 'Y', each element of S must be positive. If S is output, each element of S is a power of the radix. If S is input, each element of S should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by DGETRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). Componentwise relative backward error. This is the componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

N_ERR_BNDS Input parameter.

N_ERR_BNDS is INTEGER

Number of error bounds to return for each right hand side and each type (normwise or componentwise). See ERR_BNDS_NORM and ERR_BNDS_COMP below.

ERR_BNDS_NORM Output parameter.

ERR_BNDS_NORM is DOUBLE PRECISION

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * dlamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * dlamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * dlamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Output parameter.

ERR_BNDS_COMP is DOUBLE PRECISION

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:, N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * dlamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * dlamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * dlamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

NPARAMS Input parameter.

NPARAMS is INTEGER

Specifies the number of parameters set in PARAMS. If .LE. 0, the PARAMS array is never referenced and default values are used.

PARAMS Input and output parameter.

PARAMS is DOUBLE PRECISION

PARAMS is an array, dimension NPARAMS. Specifies algorithm parameters. If an entry is .LT. 0.0, then that entry will be filled with default value used for that parameter. Only positions up to NPARAMS are accessed; defaults are used for higher-numbered parameters.

PARAMS(LA_LINRX_ITREF_I = 1) : Whether to perform iterative refinement or not. Default: 1.0D+0 = 0.0 : No refinement is performed, and no error bounds are computed. = 1.0 : Use the double-precision refinement algorithm, possibly with doubled-single computations if the compilation environment does not support DOUBLE PRECISION. (other values are reserved for future use)

PARAMS(LA_LINRX_ITHRESH_I = 2) : Maximum number of residual computations allowed for refinement. Default: 10 Aggressive: Set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in err_bnds_norm and err_bnds_comp may no longer be trustworthy.

PARAMS(LA_LINRX_CWISE_I = 3) : Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Default: 1.0 (attempt componentwise convergence)

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. The solution to every right-hand side is guaranteed. < 0: If INFO = -i, the i-th argument had an illegal value > 0 and <= N: U(INFO, INFO) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+J: The solution corresponding to the Jth right-hand side is not guaranteed. The solutions corresponding to other right- hand sides K with K > J may not be guaranteed as well, but only the first such right-hand side is reported. If a small componentwise error is not requested (PARAMS(3) = 0.0) then the Jth right-hand side is the first with a normwise error bound that is not guaranteed (the smallest J such that ERR_BNDS_NORM(J,1) = 0.0). By default (PARAMS(3) = 1.0) the Jth right-hand side is the first with either a normwise or componentwise error bound that is not guaranteed (the smallest J such that either ERR_BNDS_NORM(J,1) = 0.0 or ERR_BNDS_COMP(J,1) = 0.0). See the definition of ERR_BNDS_NORM(:,1) and ERR_BNDS_COMP(:,1). To get information about all of the right-hand sides check ERR_BNDS_NORM or ERR_BNDS_COMP.

Related Information

For this routine in other precisions, please see *cporfsx*, *dporfsx* and *sporfsx*. It also exists with a native C interface as *LAPACKE_zporfsx*.

4.9.63 zpprfs

zpprfs improves the computed solution to a system of linear equations when the coefficient matrix is Hermitian positive definite and packed, and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine zpprfs(UPLO, N, NRHS, AP, AFP, B, LDB, X, LDX, FERR, BERR, WORK,
                  RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zpprfs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_doublecomplex_t *ap,
             const armpl_doublecomplex_t *afp, const armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, armpl_doublecomplex_t *x,
             const armpl_int_t *ldx, double *ferr, double *berr,
             armpl_doublecomplex_t *work, double *rwork, armpl_int_t *info,
             ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AP Input parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). The upper or lower triangle of the Hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

AFP Input parameter.

AFP is COMPLEX*16

AFP is an array, dimension (N*(N+1)/2). The triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H , as computed by DPPTRF/ZPPTRF, packed columnwise in a linear array in the same format as A (see AP).

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by ZPPTRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cpprfs*, *dpprfs* and *spprfs*. It also exists with a native C interface as *LAPACKE_zpprfs*.

4.9.64 zptrfs

zptrfs improves the computed solution to a system of linear equations when the coefficient matrix is Hermitian positive definite and tridiagonal, and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine zptrfs(UPLO, N, NRHS, D, E, DF, EF, B, LDB, X, LDX, FERR, BERR,
                  WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zptrfs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const double *d, const armpl_doublecomplex_t *e,
             const double *df, const armpl_doublecomplex_t *ef,
             const armpl_doublecomplex_t *b, const armpl_int_t *ldb,
             armpl_doublecomplex_t *x, const armpl_int_t *ldx, double *ferr,
             double *berr, armpl_doublecomplex_t *work, double *rwork,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the superdiagonal or the subdiagonal of the tridiagonal matrix A is stored and the form of the factorization: = 'U': E is the superdiagonal of A, and A = U^H *D*U; = 'L': E is the subdiagonal of A, and A = L*D*L^H . (The two forms are equivalent if A is real.)

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The n real diagonal elements of the tridiagonal matrix A.

E Input parameter.

E is COMPLEX*16

E is an array, dimension (N-1). The (n-1) off-diagonal elements of the tridiagonal matrix A (see UPLO).

DF Input parameter.

DF is DOUBLE PRECISION

DF is an array, dimension (N). The n diagonal elements of the diagonal matrix D from the factorization computed by ZPTTRF.

EF Input parameter.

EF is COMPLEX*16

EF is an array, dimension (N-1). The (n-1) off-diagonal elements of the unit bidiagonal factor U or L from the factorization computed by ZPTTRF (see UPLO).

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by ZPTTRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j).

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cptrfs*, *dptrfs* and *sptrfs*. It also exists with a native C interface as *LAPACKE_zptrfs*.

4.9.65 zsprfs

zsprfs improves the computed solution to a system of linear equations when the coefficient matrix is symmetric indefinite and packed, and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsprfs(UPLO, N, NRHS, AP, AFP, IPIV, B, LDB, X, LDX, FERR, BERR,
                  WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zsprfs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_doublecomplex_t *ap,
             const armpl_doublecomplex_t *afp, const armpl_int_t *ipiv,
             const armpl_doublecomplex_t *b, const armpl_int_t *ldb,
             armpl_doublecomplex_t *x, const armpl_int_t *ldx, double *ferr,
             double *berr, armpl_doublecomplex_t *work, double *rwork,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AP Input parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). The upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n.

AFP Input parameter.

AFP is COMPLEX*16

AFP is an array, dimension (N*(N+1)/2). The factored form of the matrix A. AFP contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by ZSPTRF, stored as a packed triangular matrix.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZSPTRF.

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by ZSPTRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csprfs*, *dsprfs* and *ssprfs*. It also exists with a native C interface as *LAPACKE_zsprfs*.

4.9.66 zsyrfs

zsyrfs improves the computed solution to a system of linear equations when the coefficient matrix is symmetric indefinite, and provides error bounds and backward error estimates for the solution.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsyrfs(UPLO, N, NRHS, A, LDA, AF, LDAF, IPIV, B, LDB, X, LDX, FERR,
                  BERR, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zsyrfs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_doublecomplex_t *af, const armpl_int_t *ldaf,
             const armpl_int_t *ipiv, const armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, armpl_doublecomplex_t *x,
             const armpl_int_t *ldx, double *ferr, double *berr,
             armpl_doublecomplex_t *work, double *rwork, armpl_int_t *info,
             ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX*16

AF is an array, dimension (LDAF, N). The factored form of the matrix A. AF contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by ZSYTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZSYTRF.

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by ZSYTRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csyrfs*, *dsyrfs* and *ssyrfs*. It also exists with a native C interface as *LAPACKE_zsyrfs*.

4.9.67 zsyrfsx

```
ZSYRFSX improves the computed solution to a system of linear
   equations when the coefficient matrix is symmetric indefinite, and
   provides error bounds and backward error estimates for the
   solution.  In addition to normwise error bound, the code provides
   maximum componentwise error bound if possible.  See comments for
   ERR_BNDS_NORM and ERR_BNDS_COMP for details of the error bounds.

   The original system of linear equations may have been equilibrated
   before calling this routine, as described by arguments EQUED and S
   below. In this case, the solution and error bounds returned are
   for the original unequilibrated system.                    Some optional␣
↪parameters are bundled in the PARAMS array.  These
    settings determine how refinement is performed, but often the
    defaults are acceptable.  If the defaults are acceptable, users
    can pass NPARAMS = 0 which prevents the source code from accessing
    the PARAMS argument.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zsyrfsx(UPLO, EQUED, N, NRHS, A, LDA, AF, LDAF, IPIV, S, B, LDB, X,
                   LDX, RCOND, BERR, N_ERR_BNDS, ERR_BNDS_NORM, ERR_BNDS_COMP,
                   NPARAMS, PARAMS, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zsyrfsx_(const char *uplo, const char *equed, const armpl_int_t *n,
              const armpl_int_t *nrhs, const armpl_doublecomplex_t *a,
              const armpl_int_t *lda, const armpl_doublecomplex_t *af,
              const armpl_int_t *ldaf, const armpl_int_t *ipiv,
              const double *s, const armpl_doublecomplex_t *b,
              const armpl_int_t *ldb, armpl_doublecomplex_t *x,
              const armpl_int_t *ldx, double *rcond, double *berr,
              const armpl_int_t *n_err_bnds, double *err_bnds_norm,
              double *err_bnds_comp, const armpl_int_t *nparams,
              double *params, armpl_doublecomplex_t *work, double *rwork,
              armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

EQUED Input parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done to A before calling this routine. This is needed to compute the solution and error bounds correctly. = 'N': No equilibration = 'Y': Both row and column equilibration, i.e., A has been replaced by diag(S) * A * diag(S). The right hand side B has been changed accordingly.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX*16

AF is an array, dimension (LDAF, N). The factored form of the matrix A. AF contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization A = U*D*U^T or A = L*D*L^T as computed by DSYTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by DSYTRF.

S Input and output parameter.

S is DOUBLE PRECISION

S is an array, dimension (N). The scale factors for A. If EQUED = 'Y', A is multiplied on the left and right by diag(S). S is an input argument if FACT = 'F'; otherwise, S is an output argument. If FACT = 'F' and EQUED = 'Y', each element of S must be positive. If S is output, each element of S is a power of the radix. If S is input, each element of S should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). On entry, the solution matrix X, as computed by DGETRS. On exit, the improved solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). Componentwise relative backward error. This is the componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

N_ERR_BNDS Input parameter.

N_ERR_BNDS is INTEGER

Number of error bounds to return for each right hand side and each type (normwise or componentwise). See ERR_BNDS_NORM and ERR_BNDS_COMP below.

ERR_BNDS_NORM Output parameter.

ERR_BNDS_NORM is DOUBLE PRECISION

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 “Trust/don’t trust” boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * dlamch(‘Epsilon’).

err = 2 “Guaranteed” error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * dlamch(‘Epsilon’). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * dlamch(‘Epsilon’) to determine if the error estimate is “guaranteed”. These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Output parameter.

ERR_BNDS_COMP is DOUBLE PRECISION

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:, N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * dlamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * dlamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * dlamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

See Lapack Working Note 165 for further details and extra cautions.

NPARAMS Input parameter.

NPARAMS is INTEGER

Specifies the number of parameters set in PARAMS. If .LE. 0, the PARAMS array is never referenced and default values are used.

PARAMS Input and output parameter.

PARAMS is DOUBLE PRECISION

PARAMS is an array, dimension NPARAMS. Specifies algorithm parameters. If an entry is .LT. 0.0, then that entry will be filled with default value used for that parameter. Only positions up to NPARAMS are accessed; defaults are used for higher-numbered parameters.

PARAMS(LA_LINRX_ITREF_I = 1) : Whether to perform iterative refinement or not. Default: 1.0D+0 = 0.0 : No refinement is performed, and no error bounds are computed. = 1.0 : Use the double-precision refinement algorithm, possibly with doubled-single computations if the compilation environment does not support DOUBLE PRECISION. (other values are reserved for future use)

PARAMS(LA_LINRX_ITHRESH_I = 2) : Maximum number of residual computations allowed for refinement. Default: 10 Aggressive: Set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in err_bnds_norm and err_bnds_comp may no longer be trustworthy.

PARAMS(LA_LINRX_CWISE_I = 3) : Flag determining if the code will attempt to find a solution with small componentwise relative error in the double-precision algorithm. Positive is true, 0.0 is false. Default: 1.0 (attempt componentwise convergence)

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. The solution to every right-hand side is guaranteed. < 0: If INFO = -i, the i-th argument had an illegal value > 0 and <= N: U(INFO, INFO) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned. = N+J: The solution corresponding to the Jth right-hand side is not guaranteed. The solutions corresponding to other right- hand sides K with K > J may not be guaranteed as well, but only the first such right-hand side is reported. If a small componentwise error is not requested (PARAMS(3) = 0.0) then the Jth right-hand side is the first with a normwise error bound that is not guaranteed (the smallest J such that ERR_BNDS_NORM(J,1) = 0.0). By default (PARAMS(3) = 1.0) the Jth right-hand side is the first with either a normwise or componentwise error bound that is not guaranteed (the smallest J such that either ERR_BNDS_NORM(J,1) = 0.0 or ERR_BNDS_COMP(J,1) = 0.0). See the definition of ERR_BNDS_NORM(:,1) and ERR_BNDS_COMP(:,1). To get information about all of the right-hand sides check ERR_BNDS_NORM or ERR_BNDS_COMP.

Related Information

For this routine in other precisions, please see *csyrfsx*, *dsyrfsx* and *ssyrfsx*. It also exists with a native C interface as *LAPACKE_zsyrfsx*.

4.9.68 ztbrfs

ztbrfs provides error bounds and backward error estimates for the solution to a system of linear equations with a triangular band coefficient matrix.

The solution matrix X must be computed by ZTBTRS or some other means before entering this routine. ztbrfs does not do iterative refinement because doing so cannot improve the backward error.

Syntax

Fortran specification:

```
use armpl_library

subroutine ztbrfs(UPLO, TRANS, DIAG, N, KD, NRHS, AB, LDAB, B, LDB, X, LDX,
                  FERR, BERR, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ztbrfs_(const char *uplo, const char *trans, const char *diag,
             const armpl_int_t *n, const armpl_int_t *kd,
             const armpl_int_t *nrhs, const armpl_doublecomplex_t *ab,
             const armpl_int_t *ldab, const armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, const armpl_doublecomplex_t *x,
             const armpl_int_t *ldx, double *ferr, double *berr,
             armpl_doublecomplex_t *work, double *rwork, armpl_int_t *info,
             ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose)

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals or subdiagonals of the triangular band matrix A. KD >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AB Input parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). The upper or lower triangular band matrix A, stored in the first kd+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd). If DIAG = 'U', the diagonal elements of A are not referenced and are assumed to be 1.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). The solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctbrfs*, *dtbrfs* and *stbrfs*. It also exists with a native C interface as *LAPACKE_ztbrfs*.

4.9.69 ztprfs

ztprfs provides error bounds and backward error estimates for the solution to a system of linear equations with a triangular packed coefficient matrix.

The solution matrix X must be computed by ZTPTRS or some other means before entering this routine. ztprfs does not do iterative refinement because doing so cannot improve the backward error.

Syntax

Fortran specification:

```
use armpl_library

subroutine ztprfs(UPLO, TRANS, DIAG, N, NRHS, AP, B, LDB, X, LDX, FERR, BERR,
                  WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ztprfs_(const char *uplo, const char *trans, const char *diag,
             const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_doublecomplex_t *ap, const armpl_doublecomplex_t *b,
```

(continues on next page)

(continued from previous page)

```
            const armpl_int_t *ldb, const armpl_doublecomplex_t *x,
            const armpl_int_t *ldx, double *ferr, double *berr,
            armpl_doublecomplex_t *work, double *rwork, armpl_int_t *info,
            ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose)

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

AP Input parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). The upper or lower triangular matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n. If DIAG = 'U', the diagonal elements of A are not referenced and are assumed to be 1.

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). The solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctprfs*, *dtprfs* and *stprfs*. It also exists with a native C interface as *LAPACKE_ztprfs*.

4.9.70 ztrrfs

ztrrfs provides error bounds and backward error estimates for the solution to a system of linear equations with a triangular coefficient matrix.

The solution matrix X must be computed by ZTRTRS or some other means before entering this routine. ztrrfs does not do iterative refinement because doing so cannot improve the backward error.

Syntax

Fortran specification:

```
use armpl_library

subroutine ztrrfs(UPLO, TRANS, DIAG, N, NRHS, A, LDA, B, LDB, X, LDX, FERR,
                  BERR, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ztrrfs_(const char *uplo, const char *trans, const char *diag,
             const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
```

(continues on next page)

(continued from previous page)

```
                const armpl_doublecomplex_t *b, const armpl_int_t *ldb,
                const armpl_doublecomplex_t *x, const armpl_int_t *ldx,
                double *ferr, double *berr, armpl_doublecomplex_t *work,
                double *rwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose)

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). The right hand side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

X Input parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). The solution matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N).

FERR Output parameter.

FERR is DOUBLE PRECISION

FERR is an array, dimension (NRHS). The estimated forward error bound for each solution vector X(j) (the j-th column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctrrfs*, *dtrrfs* and *strrfs*. It also exists with a native C interface as *LAPACKE_ztrrfs*.

4.10 LAPACK singular value decompositions routines

4.10.1 cbdsqr

cbdsqr computes the singular values and, optionally, the right and/or left singular vectors from the singular value decomposition (SVD) of a real N-by-N (upper or lower) bidiagonal matrix B using the implicit zero-shift QR algorithm. The SVD of B has the form

```
B = Q * S * P**H
```

where S is the diagonal matrix of singular values, Q is an orthogonal matrix of left singular vectors, and P is an orthogonal matrix of right singular vectors. If left singular vectors are requested, this subroutine actually returns U*Q instead of Q, and, if right singular vectors are requested, this subroutine returns P^H *VT instead of P^H , for given complex input matrices U and VT. When U and VT are the unitary matrices that reduce a general matrix A to bidiagonal form: A = U*B*VT, as computed by CGEBRD, then

```
A = (U*Q) * S * (P**H*VT)
```

is the SVD of A. Optionally, the subroutine may also compute Q^H *C for a given complex input matrix C.

See "Computing Small Singular Values of Bidiagonal Matrices With Guaranteed High Relative Accuracy," by J. Demmel and W. Kahan, LAPACK Working Note #3 (or SIAM J. Sci. Statist. Comput. vol. 11, no. 5, pp. 873-912, Sept 1990) and "Accurate singular values and differential qd algorithms," by B. Parlett and V. Fernando, Technical Report CPAM-554, Mathematics Department, University of California at Berkeley, July 1992 for a detailed description of the algorithm.

Syntax

Fortran specification:

```
use armpl_library

subroutine cbdsqr(UPLO, N, NCVT, NRU, NCC, D, E, VT, LDVT, U, LDU, C, LDC,
                  RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cbdsqr_(const char *uplo, const armpl_int_t *n, const armpl_int_t *ncvt,
             const armpl_int_t *nru, const armpl_int_t *ncc, float *d,
             float *e, armpl_singlecomplex_t *vt, const armpl_int_t *ldvt,
             armpl_singlecomplex_t *u, const armpl_int_t *ldu,
             armpl_singlecomplex_t *c, const armpl_int_t *ldc, float *rwork,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': B is upper bidiagonal; = 'L': B is lower bidiagonal.

N Input parameter.

N is INTEGER

The order of the matrix B. N >= 0.

NCVT Input parameter.

NCVT is INTEGER

The number of columns of the matrix VT. NCVT >= 0.

NRU Input parameter.

NRU is INTEGER

The number of rows of the matrix U. NRU >= 0.

NCC Input parameter.

NCC is INTEGER

The number of columns of the matrix C. NCC >= 0.

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry, the n diagonal elements of the bidiagonal matrix B. On exit, if INFO=0, the singular values of B in decreasing order.

E Input and output parameter.

E is REAL

E is an array, dimension (N-1). On entry, the N-1 offdiagonal elements of the bidiagonal matrix B. On exit, if INFO = 0, E is destroyed; if INFO > 0, D and E will contain the diagonal and superdiagonal elements of a bidiagonal matrix orthogonally equivalent to the one given as input.

VT Input and output parameter.

VT is COMPLEX

VT is an array, dimension (LDVT, NCVT). On entry, an N-by-NCVT matrix VT. On exit, VT is overwritten by P^H * VT. Not referenced if NCVT = 0.

LDVT Input parameter.

LDVT is INTEGER

The leading dimension of the array VT. LDVT >= max(1, N) if NCVT > 0; LDVT >= 1 if NCVT = 0.

U Input and output parameter.

U is COMPLEX

U is an array, dimension (LDU, N). On entry, an NRU-by-N matrix U. On exit, U is overwritten by U * Q. Not referenced if NRU = 0.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U. LDU >= max(1, NRU).

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, NCC). On entry, an N-by-NCC matrix C. On exit, C is overwritten by Q^H * C. Not referenced if NCC = 0.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, N) if NCC > 0; LDC >=1 if NCC = 0.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (4*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: If INFO = -i, the i-th argument had an illegal value > 0: the algorithm did not converge; D and E contain the elements of a bidiagonal matrix which is orthogonally similar to the input matrix B; if INFO = i, i elements of E have not converged to zero.

Related Information

For this routine in other precisions, please see *dbdsqr*, *sbdsqr* and *zbdsqr*. It also exists with a native C interface as *LAPACKE_cbdsqr*.

4.10.2 cgbbrd

`cgbbrd` reduces a complex general m-by-n band matrix A to real upper bidiagonal form B by a unitary transformation: Q^H * A * P = B.

The routine computes B, and optionally forms Q or P^H , or computes Q^H *C for a given matrix C.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgbbrd(VECT, M, N, NCC, KL, KU, AB, LDAB, D, E, Q, LDQ, PT, LDPT,
                  C, LDC, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgbbrd_(const char *vect, const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *ncc, const armpl_int_t *kl,
             const armpl_int_t *ku, armpl_singlecomplex_t *ab,
             const armpl_int_t *ldab, float *d, float *e,
             armpl_singlecomplex_t *q, const armpl_int_t *ldq,
             armpl_singlecomplex_t *pt, const armpl_int_t *ldpt,
             armpl_singlecomplex_t *c, const armpl_int_t *ldc,
             armpl_singlecomplex_t *work, float *rwork, armpl_int_t *info,
             ... );
```

Parameters

VECT Input parameter.

`VECT` is `CHARACTER*1`

Specifies whether or not the matrices `Q` and P^H are to be formed. = 'N': do not form `Q` or P^H ; = 'Q': form `Q` only; = 'P': form P^H only; = 'B': form both.

M Input parameter.

`M` is `INTEGER`

The number of rows of the matrix A. `M` >= 0.

N Input parameter.

`N` is `INTEGER`

The number of columns of the matrix A. `N` >= 0.

NCC Input parameter.

`NCC` is `INTEGER`

The number of columns of the matrix `C`. `NCC` >= 0.

KL Input parameter.

`KL` is `INTEGER`

The number of subdiagonals of the matrix A. `KL` >= 0.

KU Input parameter.

`KU` is `INTEGER`

The number of superdiagonals of the matrix A. KU >= 0.

AB Input and output parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). On entry, the m-by-n band matrix A, stored in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(ku+1+i-j,j) = A(i,j) for max(1,j-ku)<=i<=min(m,j+kl). On exit, A is overwritten by values generated during the reduction.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array A. LDAB >= KL+KU+1.

D Output parameter.

D is REAL

D is an array, dimension (min(M, N)). The diagonal elements of the bidiagonal matrix B.

E Output parameter.

E is REAL

E is an array, dimension (min(M, N)-1). The superdiagonal elements of the bidiagonal matrix B.

Q Output parameter.

Q is COMPLEX

Q is an array, dimension (LDQ, M). If VECT = 'Q' or 'B', the m-by-m unitary matrix Q. If VECT = 'N' or 'P', the array Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, M) if VECT = 'Q' or 'B'; LDQ >= 1 otherwise.

PT Output parameter.

PT is COMPLEX

PT is an array, dimension (LDPT, N). If VECT = 'P' or 'B', the n-by-n unitary matrix P'. If VECT = 'N' or 'Q', the array PT is not referenced.

LDPT Input parameter.

LDPT is INTEGER

The leading dimension of the array PT. LDPT >= max(1, N) if VECT = 'P' or 'B'; LDPT >= 1 otherwise.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, NCC). On entry, an m-by-ncc matrix C. On exit, C is overwritten by Q^H *C. C is not referenced if NCC = 0.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M) if NCC > 0; LDC >= 1 if NCC = 0.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (max(M, N)) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (max(M, N)) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *dgbbrd*, *sgbbrd* and *zgbbrd*. It also exists with a native C interface as *LAPACKE_cgbbrd*.

4.10.3 cgebrd

cgebrd reduces a general complex M-by-N matrix A to upper or lower bidiagonal form B by a unitary transformation: $Q^H * A * P = B$.

If m >= n, B is upper bidiagonal; if m < n, B is lower bidiagonal.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgebrd(M, N, A, LDA, D, E, TAUQ, TAUP, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgebrd_(const armpl_int_t *m, const armpl_int_t *n,
             armpl_singlecomplex_t *a, const armpl_int_t *lda, float *d,
             float *e, armpl_singlecomplex_t *tauq,
             armpl_singlecomplex_t *taup, armpl_singlecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows in the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns in the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the M-by-N general matrix to be reduced. On exit, if m >= n, the diagonal and the first superdiagonal are overwritten with the upper bidiagonal matrix B; the elements below the diagonal, with the array TAUQ, represent the unitary matrix Q as a product of elementary reflectors, and the elements above the first superdiagonal, with the array TAUP, represent the unitary matrix P as a product of elementary reflectors; if m < n, the diagonal and the first subdiagonal are overwritten with the lower bidiagonal matrix B; the elements below the first subdiagonal, with the array TAUQ, represent the

unitary matrix Q as a product of elementary reflectors, and the elements above the diagonal, with the array TAUP, represent the unitary matrix P as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

D Output parameter.

D is REAL

D is an array, dimension (min(M, N)). The diagonal elements of the bidiagonal matrix B: D(i) = A(i,i).

E Output parameter.

E is REAL

E is an array, dimension (min(M, N)-1). The off-diagonal elements of the bidiagonal matrix B: if m >= n, E(i) = A(i,i+1) for i = 1,2,...,n-1; if m < n, E(i) = A(i+1,i) for i = 1,2,...,m-1.

TAUQ Output parameter.

TAUQ is COMPLEX

TAUQ is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors which represent the unitary matrix Q. See Further Details.

TAUP Output parameter.

TAUP is COMPLEX

TAUP is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors which represent the unitary matrix P. See Further Details.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= max(1, M, N). For optimum performance LWORK >= (M+N)*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *dgebrd*, *sgebrd* and *zgebrd*. It also exists with a native C interface as *LAPACKE_cgebrd*.

4.10.4 cgejsv

cgejsv computes the singular value decomposition (SVD) of a complex M-by-N matrix [A], where M >= N. The SVD of [A] is written as

```
[A] = [U] * [SIGMA] * [V]^*,
```

where [SIGMA] is an N-by-N (M-by-N) matrix which is zero except for its N diagonal elements, [U] is an M-by-N (or M-by-M) unitary matrix, and [V] is an N-by-N unitary matrix. The diagonal elements of [SIGMA] are the singular values of [A]. The columns of [U] and [V] are the left and the right singular vectors of [A], respectively. The matrices [U] and [V] are computed and stored in the arrays U and V, respectively. The diagonal of [SIGMA] is computed and stored in the array SVA.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgejsv(JOBA, JOBU, JOBV, JOBR, JOBT, JOBP, M, N, A, LDA, SVA, U,
                  LDU, V, LDV, CWORK, LWORK, RWORK, LRWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgejsv_(const char *joba, const char *jobu, const char *jobv,
             const char *jobr, const char *jobt, const char *jobp,
             const armpl_int_t *m, const armpl_int_t *n,
             armpl_singlecomplex_t *a, const armpl_int_t *lda, float *sva,
             armpl_singlecomplex_t *u, const armpl_int_t *ldu,
             armpl_singlecomplex_t *v, const armpl_int_t *ldv,
             armpl_singlecomplex_t *cwork, const armpl_int_t *lwork,
             float *rwork, const armpl_int_t *lrwork, armpl_int_t *iwork,
             armpl_int_t *info, ... );
```

Parameters

JOBA Input parameter.

JOBA is CHARACTER*1

Specifies the level of accuracy: = 'C': This option works well (high relative accuracy) if A = B * D, with well-conditioned B and arbitrary diagonal matrix D. The accuracy cannot be spoiled by COLUMN scaling. The accuracy of the computed output depends on the condition of B, and the procedure aims at the best theoretical accuracy. The relative error max_{i=1:N}|d sigma_i| / sigma_i is bounded by f(M, N)*epsilon* cond(B), independent of D. The input matrix is preprocessed with the QRF with column pivoting. This initial preprocessing and preconditioning by a rank revealing QR factorization is common for all values of JOBA. Additional actions are specified as follows: = 'E': Computation as with 'C' with an additional estimate of the condition number of B. It provides a realistic error bound. = 'F': If A = D1 * C * D2 with ill-conditioned diagonal scalings D1, D2, and well-conditioned matrix C, this option gives higher accuracy than the 'C' option. If the structure of the input matrix is not known, and relative accuracy is desirable, then this option is advisable. The input matrix A is preprocessed with QR factorization with FULL (row and column) pivoting. = 'G' Computation as with 'F' with an additional estimate of the condition number of B, where A=B*D. If A has heavily weighted rows, then using this condition number gives too pessimistic error bound. = 'A': Small singular values are not well determined by the data and are considered as noisy; the matrix is treated as numerically rank defficient. The error in the computed singular values is bounded by f(m,n)*epsilon*||A||. The computed SVD A = U * S * V^* restores A up to f(m,n)*epsilon*||A||. This gives the procedure the licence to discard (set to zero) all singular values below N*epsilon*||A||. = 'R': Similar as

in ‘A’. Rank revealing property of the initial QR factorization is used do reveal (using triangular factor) a gap sigma_{r+1} < epsilon * sigma_r in which case the numerical RANK is declared to be r. The SVD is computed with absolute error bounds, but more accurately than with ‘A’.

JOBU Input parameter.

JOBU is CHARACTER*1

Specifies whether to compute the columns of U: = ‘U’: N columns of U are returned in the array U. = ‘F’: full set of M left sing. vectors is returned in the array U. = ‘W’: U may be used as workspace of length M*N. See the description of U. = ‘N’: U is not computed.

JOBV Input parameter.

JOBV is CHARACTER*1

Specifies whether to compute the matrix V: = ‘V’: N columns of V are returned in the array V; Jacobi rotations are not explicitly accumulated. = ‘J’: N columns of V are returned in the array V, but they are computed as the product of Jacobi rotations, if JOBT .EQ. ‘N’. = ‘W’: V may be used as workspace of length N*N. See the description of V. = ‘N’: V is not computed.

JOBR Input parameter.

JOBR is CHARACTER*1

Specifies the RANGE for the singular values. Issues the licence to set to zero small positive singular values if they are outside specified range. If A .NE. 0 is scaled so that the largest singular value of c* A is around SQRT(BIG), BIG=SLAMCH(‘O’), then JOBR issues the licence to kill columns of A whose norm in c* A is less than SQRT(SFMIN) (for JOBR.EQ.’R’), or less than SMALL=SFMIN/EPSLN, where SFMIN=SLAMCH(‘S’), EPSLN=SLAMCH(‘E’). = ‘N’: Do not kill small columns of c*A. This option assumes that BLAS and QR factorizations and triangular solvers are implemented to work in that range. If the condition of A is greater than BIG, use CGESVJ. = ‘R’: RESTRICTED range for sigma(c*A) is [SQRT(SFMIN), SQRT(BIG)] (roughly, as described above). This option is recommended. ========== For computing the singular values in the FULL range [SFMIN,BIG] use CGESVJ.

JOBT Input parameter.

JOBT is CHARACTER*1

If the matrix is square then the procedure may determine to use transposed A if A^* seems to be better with respect to convergence. If the matrix is not square, JOBT is ignored. The decision is based on two values of entropy over the adjoint orbit of A^* * A. See the descriptions of WORK(6) and WORK(7). = ‘T’: transpose if entropy test indicates possibly faster convergence of Jacobi process if A^* is taken as input. If A is replaced with A^*, then the row pivoting is included automatically. = ‘N’: do not speculate. The option ‘T’ can be used to compute only the singular values, or the full SVD (U, SIGMA and V). For only one set of singular vectors (U or V), the caller should provide both U and V, as one of the matrices is used as workspace if the matrix A is transposed. The implementer can easily remove this constraint and make the code more complicated. See the descriptions of U and V. In general, this option is considered experimental, and ‘N’; should be preferred. This is subject to changes in the future.

JOBP Input parameter.

JOBP is CHARACTER*1

Issues the licence to introduce structured perturbations to drown denormalized numbers. This licence should be active if the denormals are poorly implemented, causing slow computation, especially in cases of fast convergence (!). For details see [1,2]. For the sake of simplicity, this perturbations are included only when the full SVD or only the singular values are requested. The implementer/user can easily add the perturbation for the cases of computing one set of singular vectors. = ‘P’: introduce perturbation = ‘N’: do not perturb

M Input parameter.

M is INTEGER

The number of rows of the input matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the input matrix A. M >= N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

SVA Output parameter.

SVA is REAL

SVA is an array, dimension (N). On exit, - For WORK(1)/WORK(2) = ONE: The singular values of A. During the computation SVA contains Euclidean column norms of the iterated matrices in the array A. - For WORK(1) .NE. WORK(2): The singular values of A are (WORK(1)/WORK(2)) * SVA(1:N). This factored form is used if sigma_max(A) overflows or if small singular values have been saved from underflow by scaling the input matrix A. - If JOBR='R' then some of the singular values may be returned as exact zeros obtained by "set to zero" because they are below the numerical rank threshold or are denormalized numbers.

U Output parameter.

U is COMPLEX

U is an array, dimension (LDU, N) or (LDU, M). If JOBU = 'U', then U contains on exit the M-by-N matrix of the left singular vectors. If JOBU = 'F', then U contains on exit the M-by-M matrix of the left singular vectors, including an ONB of the orthogonal complement of the Range(A). If JOBU = 'W' .AND. (JOBV.EQ.'V' .AND. JOBT.EQ.'T' .AND. M.EQ.N), then U is used as workspace if the procedure replaces A with A^*. In that case, [V] is computed in U as left singular vectors of A^* and then copied back to the V array. This 'W' option is just a reminder to the caller that in this case U is reserved as workspace of length N*N. If JOBU = 'N' U is not referenced, unless JOBT='T'.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U, LDU >= 1. IF JOBU = 'U' or 'F' or 'W', then LDU >= M.

V Output parameter.

V is COMPLEX

V is an array, dimension (LDV, N). If JOBV = 'V', 'J' then V contains on exit the N-by-N matrix of the right singular vectors; If JOBV = 'W', AND (JOBU.EQ.'U' AND JOBT.EQ.'T' AND M.EQ.N), then V is used as workspace if the pprocedure replaces A with A^*. In that case, [U] is computed in V as right singular vectors of A^* and then copied back to the U array. This 'W' option is just a reminder to the caller that in this case V is reserved as workspace of length N*N. If JOBV = 'N' V is not referenced, unless JOBT='T'.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V, LDV >= 1. If JOBV = 'V' or 'J' or 'W', then LDV >= N.

CWORK Output parameter.

CWORK is COMPLEX

CWORK is an array, dimension (MAX(2, LWORK)). If the call to CGEJSV is a workspace query (indicated by LWORK=-1 or LRWORK=-1), then on exit CWORK(1) contains the required length of CWORK for the job parameters used in the call.

LWORK Input parameter.

LWORK is INTEGER

Length of CWORK to confirm proper allocation of workspace. LWORK depends on the job:

1. If only SIGMA is needed (JOBU.EQ.'N', JOBV.EQ.'N') and 1.1 .. no scaled condition estimate required (JOBA.NE.'E'.AND.JOBA.NE.'G'): LWORK >= 2*N+1. This is the minimal requirement. ->> For optimal performance (blocked code) the optimal value is LWORK >= N + (N+1)*NB. Here NB is the optimal block size for CGEQP3 and CGEQRF. In general, optimal LWORK is computed as LWORK >= max(N+LWORK(CGEQP3),N+LWORK(CGEQRF), LWORK(CGESVJ)). 1.2. .. an estimate of the scaled condition number of A is required (JOBA='E', or 'G'). In this case, LWORK the minimal requirement is LWORK >= N* N + 2*N. ->> For optimal performance (blocked code) the optimal value is LWORK >= max(N+(N+1)*NB, N*N+2*N)=N**2+2*N. In general, the optimal length LWORK is computed as LWORK >= max(N+LWORK(CGEQP3),N+LWORK(CGEQRF), LWORK(CGESVJ), N*N+LWORK(CPOCON)). 2. If SIGMA and the right singular vectors are needed (JOBV.EQ.'V'), (JOBU.EQ.'N') 2.1 .. no scaled condition estimate requested (JOBE.EQ.'N'): -> the minimal requirement is LWORK >= 3*N. -> For optimal performance, LWORK >= max(N+(N+1)*NB, 2*N+N*NB)=2*N+N*NB, where NB is the optimal block size for CGEQP3, CGEQRF, CGELQ, CUNMLQ. In general, the optimal length LWORK is computed as LWORK >= max(N+LWORK(CGEQP3), N+LWORK(CGESVJ), N+LWORK(CGELQF), 2*N+LWORK(CGEQRF), N+LWORK(CUNMLQ)). 2.2 .. an estimate of the scaled condition number of A is required (JOBA='E', or 'G'). -> the minimal requirement is LWORK >= 3*N. -> For optimal performance, LWORK >= max(N+(N+1)*NB, 2*N,2*N+N*NB)=2*N+N*NB, where NB is the optimal block size for CGEQP3, CGEQRF, CGELQ, CUNMLQ. In general, the optimal length LWORK is computed as LWORK >= max(N+LWORK(CGEQP3), LWORK(CPOCON), N+LWORK(CGESVJ), N+LWORK(CGELQF), 2*N+LWORK(CGEQRF), N+LWORK(CUNMLQ)). 3. If SIGMA and the left singular vectors are needed 3.1 .. no scaled condition estimate requested (JOBE.EQ.'N'): -> the minimal requirement is LWORK >= 3*N. -> For optimal performance: if JOBU.EQ.'U' :: LWORK >= max(3*N, N+(N+1)*NB, 2*N+N*NB)=2*N+N*NB, where NB is the optimal block size for CGEQP3, CGEQRF, CUNMQR. In general, the optimal length LWORK is computed as LWORK >= max(N+LWORK(CGEQP3), 2*N+LWORK(CGEQRF), N+LWORK(CUNMQR)). 3.2 .. an estimate of the scaled condition number of A is required (JOBA='E', or 'G'). -> the minimal requirement is LWORK >= 3*N. -> For optimal performance: if JOBU.EQ.'U' :: LWORK >= max(3*N, N+(N+1)*NB, 2*N+N*NB)=2*N+N*NB, where NB is the optimal block size for CGEQP3, CGEQRF, CUNMQR. In general, the optimal length LWORK is computed as LWORK >= max(N+LWORK(CGEQP3),N+LWORK(CPOCON), 2*N+LWORK(CGEQRF), N+LWORK(CUNMQR)).

4. If the full SVD is needed: (JOBU.EQ.'U' or JOBU.EQ.'F') and 4.1. if JOBV.EQ.'V' the minimal requirement is LWORK >= 5*N+2*N*N. 4.2. if JOBV.EQ.'J' the minimal requirement is LWORK >= 4*N+N*N. In both cases, the allocated CWORK can accommodate blocked runs of CGEQP3, CGEQRF, CGELQF, CUNMQR, CUNMLQ.

If the call to CGEJSV is a workspace query (indicated by LWORK=-1 or LRWORK=-1), then on exit CWORK(1) contains the optimal and CWORK(2) contains the minimal length of CWORK for the job parameters used in the call.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (MAX(7, LWORK)). On exit, RWORK(1) = Determines the scaling factor SCALE = RWORK(2) / RWORK(1) such that SCALE*SVA(1:N) are the computed singular values of A. (See the description of SVA().) RWORK(2) = See the description of RWORK(1). RWORK(3) = SCONDA is an estimate for the condition number of column equilibrated A. (If JOBA .EQ. 'E' or 'G') SCONDA is an estimate of SQRT(||(R^* * R)^(-1)||_1). It is computed using SPOCON. It holds N^(-1/4) * SCONDA <= ||R^(-1)||_2 <= N^(1/4) * SCONDA where R is the triangular factor from the QRF of A. However, if R is truncated and the numerical rank is determined to be strictly smaller than N, SCONDA is returned as -1, thus indicating that the smallest singular values might be lost.

If full SVD is needed, the following two condition numbers are useful for the analysis of the algorithm. They are provied for a developer/implementer who is familiar with the details of the method.

RWORK(4) = an estimate of the scaled condition number of the triangular factor in the first QR factorization. RWORK(5) = an estimate of the scaled condition number of the triangular factor in the second QR factorization. The following two parameters are computed if JOBT .EQ. 'T'. They are provided for a developer/implementer who is familiar with the details of the method. RWORK(6) = the entropy of A^* * A :: this is the Shannon entropy of diag(A^* * A) / Trace(A^* * A) taken as point in the probability simplex. RWORK(7) = the entropy of A * A^*. (See the description of RWORK(6).) If the call to CGEJSV is a workspace query (indicated by LWORK=-1 or LRWORK=-1), then on exit RWORK(1) contains the required length of RWORK for the job parameters used in the call.

LRWORK Input parameter.

LRWORK is INTEGER

Length of RWORK to confirm proper allocation of workspace. LRWORK depends on the job:

1. If only the singular values are requested i.e. if LSAME(JOBU,'N') .AND. LSAME(JOBV,'N') then: 1.1. If LSAME(JOBT,'T') .OR. LSAME(JOBA,'F') .OR. LSAME(JOBA,'G'), then: LRWORK = max(7, 2 * M). 1.2. Otherwise, LRWORK = max(7, N). 2. If singular values with the right singular vectors are requested i.e. if (LSAME(JOBV,'V').OR.LSAME(JOBV,'J')) .AND. .NOT.(LSAME(JOBU,'U').OR.LSAME(JOBU,'F')) then: 2.1. If LSAME(JOBT,'T') .OR. LSAME(JOBA,'F') .OR. LSAME(JOBA,'G'), then LRWORK = max(7, 2 * M). 2.2. Otherwise, LRWORK = max(7, N). 3. If singular values with the left singular vectors are requested, i.e. if (LSAME(JOBU,'U').OR.LSAME(JOBU,'F')) .AND. .NOT.(LSAME(JOBV,'V').OR.LSAME(JOBV,'J')) then: 3.1. If LSAME(JOBT,'T') .OR. LSAME(JOBA,'F') .OR. LSAME(JOBA,'G'), then LRWORK = max(7, 2 * M). 3.2. Otherwise, LRWORK = max(7, N). 4. If singular values with both the left and the right singular vectors are requested, i.e. if (LSAME(JOBU,'U').OR.LSAME(JOBU,'F')) .AND. (LSAME(JOBV,'V').OR.LSAME(JOBV,'J')) then: 4.1. If LSAME(JOBT,'T') .OR. LSAME(JOBA,'F') .OR. LSAME(JOBA,'G'), then LRWORK = max(7, 2 * M). 4.2. Otherwise, LRWORK = max(7, N).

If, on entry, LRWORK = -1 or LWORK=-1, a workspace query is assumed and the length of RWORK is returned in RWORK(1).

IWORK Output parameter.

IWORK is INTEGER array, of dimension at least 4, that further depends

on the job:

1. If only the singular values are requested then: If (LSAME(JOBT,'T') .OR. LSAME(JOBA,'F') .OR. LSAME(JOBA,'G')) then the length of IWORK is N+M; otherwise the length of IWORK is N. 2. If the singular values and the right singular vectors are requested then: If (LSAME(JOBT,'T') .OR. LSAME(JOBA,'F') .OR. LSAME(JOBA,'G')) then the length of IWORK is N+M; otherwise the length of IWORK is N. 3. If the singular values and the left singular vectors are requested then: If (LSAME(JOBT,'T') .OR. LSAME(JOBA,'F') .OR. LSAME(JOBA,'G')) then the length of IWORK is N+M; otherwise the length of IWORK is N. 4. If the singular values with both the left and the right singular vectors are requested, then: 4.1. If LSAME(JOBV,'J') the length of IWORK is determined as follows: If (LSAME(JOBT,'T') .OR. LSAME(JOBA,'F') .OR. LSAME(JOBA,'G')) then the length of IWORK is N+M; otherwise the length of IWORK is N. 4.2. If LSAME(JOBV,'V') the length of IWORK is determined as follows: If (LSAME(JOBT,'T') .OR. LSAME(JOBA,'F') .OR. LSAME(JOBA,'G')) then the length of IWORK is 2*N+M; otherwise the length of IWORK is 2*N.

On exit, IWORK(1) = the numerical rank determined after the initial QR factorization with pivoting. See the descriptions of JOBA and JOBR. IWORK(2) = the number of the computed nonzero singular values IWORK(3) = if nonzero, a warning message: If IWORK(3).EQ.1 then some of the column norms of A were denormalized floats. The requested high accuracy is not warranted by the data. IWORK(4) = 1 or -1. If IWORK(4) .EQ. 1, then the procedure used A^* to do the job as specified by the JOB parameters. If the call to CGEJSV is a workspace query (indicated by LWORK .EQ. -1 and LRWORK .EQ. -1), then on exit IWORK(1) contains the required length of IWORK for the job parameters used in the call.

INFO Output parameter.

INFO is INTEGER

< 0 : if INFO = -i, then the i-th argument had an illegal value. = 0 : successful exit; > 0 : CGEJSV did not converge in the maximal allowed number of sweeps. The computed values may be inaccurate.

Related Information

For this routine in other precisions, please see *dgejsv*, *sgejsv* and *zgejsv*. It also exists with a native C interface as *LAPACKE_cgejsv*.

4.10.5 cgesdd

`cgesdd` computes the singular value decomposition (SVD) of a complex M-by-N matrix A, optionally computing the left and/or right singular vectors, by using divide-and-conquer method. The SVD is written

```
A = U * SIGMA * conjugate-transpose(V)
```

where SIGMA is an M-by-N matrix which is zero except for its min(m,n) diagonal elements, U is an M-by-M unitary matrix, and V is an N-by-N unitary matrix. The diagonal elements of SIGMA are the singular values of A; they are real and non-negative, and are returned in descending order. The first min(m,n) columns of U and V are the left and right singular vectors of A.

Note that the routine returns VT = V^H , not V.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgesdd(JOBZ, M, N, A, LDA, S, U, LDU, VT, LDVT, WORK, LWORK, RWORK,
                  IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgesdd_(const char *jobz, const armpl_int_t *m, const armpl_int_t *n,
             armpl_singlecomplex_t *a, const armpl_int_t *lda, float *s,
             armpl_singlecomplex_t *u, const armpl_int_t *ldu,
             armpl_singlecomplex_t *vt, const armpl_int_t *ldvt,
             armpl_singlecomplex_t *work, const armpl_int_t *lwork,
             float *rwork, armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

`JOBZ` is `CHARACTER*1`

Specifies options for computing all or part of the matrix U: = 'A': all `M` columns of `U` and all `N` rows of V^H are returned in the arrays `U` and VT; = 'S': the first min(`M`, `N`) columns of `U` and the first min(`M`, `N`) rows of V^H are returned in the arrays U and VT; = 'O': If `M` >= `N`, the first `N` columns of `U` are overwritten in the array `A` and all rows of V^H are returned in the array VT; otherwise, all columns of `U` are returned in the array `U` and the first `M` rows of V^H are overwritten in the array A; = '`N`': no columns of `U` or rows of V^H are computed.

M Input parameter.

`M` is `INTEGER`

The number of rows of the input matrix `A`. `M` >= 0.

N Input parameter.

N is INTEGER

The number of columns of the input matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, if JOBZ = 'O', A is overwritten with the first N columns of U (the left singular vectors, stored columnwise) if M >= N; A is overwritten with the first M rows of V^H (the right singular vectors, stored rowwise) otherwise. if JOBZ .ne. 'O', the contents of A are destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

S Output parameter.

S is REAL

S is an array, dimension (min(M, N)). The singular values of A, sorted so that S(i) >= S(i+1).

U Output parameter.

U is COMPLEX

U is an array, dimension (LDU,UCOL). UCOL = M if JOBZ = 'A' or JOBZ = 'O' and M < N; UCOL = min(M, N) if JOBZ = 'S'. If JOBZ = 'A' or JOBZ = 'O' and M < N, U contains the M-by-M unitary matrix U; if JOBZ = 'S', U contains the first min(M, N) columns of U (the left singular vectors, stored columnwise); if JOBZ = 'O' and M >= N, or JOBZ = 'N', U is not referenced.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U. LDU >= 1; if JOBZ = 'S' or 'A' or JOBZ = 'O' and M < N, LDU >= M.

VT Output parameter.

VT is COMPLEX

VT is an array, dimension (LDVT, N). If JOBZ = 'A' or JOBZ = 'O' and M >= N, VT contains the N-by-N unitary matrix V^H ; if JOBZ = 'S', VT contains the first min(M, N) rows of V^H (the right singular vectors, stored rowwise); if JOBZ = 'O' and M < N, or JOBZ = 'N', VT is not referenced.

LDVT Input parameter.

LDVT is INTEGER

The leading dimension of the array VT. LDVT >= 1; if JOBZ = 'A' or JOBZ = 'O' and M >= N, LDVT >= N; if JOBZ = 'S', LDVT >= min(M, N).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= 1. If LWORK = -1, a workspace query is assumed. The optimal size for the WORK array is calculated and stored in WORK(1), and no other work except argument checking is performed.

Let mx = max(M, N) and mn = min(M, N). If JOBZ = 'N', LWORK >= 2*mn + mx. If JOBZ = 'O', LWORK >= 2*mn*mn + 2*mn + mx. If JOBZ = 'S', LWORK >= mn*mn + 3*mn. If JOBZ = 'A', LWORK >= mn*mn + 2*mn + mx. These are not tight minimums in all cases; see comments inside code. For good performance, LWORK should generally be larger; a query is recommended.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (MAX(1,LRWORK)). Let mx = max(M, N) and mn = min(M, N). If JOBZ = 'N', LRWORK >= 5*mn (LAPACK <= 3.6 needs 7*mn); else if mx >> mn, LRWORK >= 5*mn*mn + 5*mn; else LRWORK >= max(5*mn*mn + 5*mn, 2*mx*mn + 2*mn*mn + mn).

IWORK Output parameter.

IWORK is INTEGER array, dimension (8*min(M, N))

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: The updating process of SBDSDC did not converge.

Related Information

For this routine in other precisions, please see *dgesdd*, *sgesdd* and *zgesdd*. It also exists with a native C interface as *LAPACKE_cgesdd*.

4.10.6 cgesvd

cgesvd computes the singular value decomposition (SVD) of a complex M-by-N matrix A, optionally computing the left and/or right singular vectors. The SVD is written

```
A = U * SIGMA * conjugate-transpose(V)
```

where SIGMA is an M-by-N matrix which is zero except for its min(m,n) diagonal elements, U is an M-by-M unitary matrix, and V is an N-by-N unitary matrix. The diagonal elements of SIGMA are the singular values of A; they are real and non-negative, and are returned in descending order. The first min(m,n) columns of U and V are the left and right singular vectors of A.

Note that the routine returns V^H , not V.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgesvd(JOBU, JOBVT, M, N, A, LDA, S, U, LDU, VT, LDVT, WORK, LWORK,
                  RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgesvd_(const char *jobu, const char *jobvt, const armpl_int_t *m,
             const armpl_int_t *n, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, float *s, armpl_singlecomplex_t *u,
             const armpl_int_t *ldu, armpl_singlecomplex_t *vt,
             const armpl_int_t *ldvt, armpl_singlecomplex_t *work,
```

(continues on next page)

(continued from previous page)

```
            const armpl_int_t *lwork, float *rwork, armpl_int_t *info,
            ... );
```

Parameters

JOBU Input parameter.

JOBU is CHARACTER*1

Specifies options for computing all or part of the matrix U: = 'A': all M columns of U are returned in array U: = 'S': the first min(m,n) columns of U (the left singular vectors) are returned in the array U; = 'O': the first min(m,n) columns of U (the left singular vectors) are overwritten on the array A; = 'N': no columns of U (no left singular vectors) are computed.

JOBVT Input parameter.

JOBVT is CHARACTER*1

Specifies options for computing all or part of the matrix V^H : = 'A': all N rows of V^H are returned in the array VT; = 'S': the first min(m,n) rows of V^H (the right singular vectors) are returned in the array VT; = 'O': the first min(m,n) rows of V^H (the right singular vectors) are overwritten on the array A; = 'N': no rows of V^H (no right singular vectors) are computed.

JOBVT and JOBU cannot both be 'O'.

M Input parameter.

M is INTEGER

The number of rows of the input matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the input matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, if JOBU = 'O', A is overwritten with the first min(m,n) columns of U (the left singular vectors, stored columnwise); if JOBVT = 'O', A is overwritten with the first min(m,n) rows of V^H (the right singular vectors, stored rowwise); if JOBU .ne. 'O' and JOBVT .ne. 'O', the contents of A are destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

S Output parameter.

S is REAL

S is an array, dimension (min(M, N)). The singular values of A, sorted so that S(i) >= S(i+1).

U Output parameter.

U is COMPLEX

U is an array, dimension (LDU,UCOL). (LDU, M) if JOBU = 'A' or (LDU,min(M, N)) if JOBU = 'S'. If JOBU = 'A', U contains the M-by-M unitary matrix U; if JOBU = 'S', U contains the first min(m,n) columns of U (the left singular vectors, stored columnwise); if JOBU = 'N' or 'O', U is not referenced.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U. LDU >= 1; if JOBU = 'S' or 'A', LDU >= M.

VT Output parameter.

VT is COMPLEX

VT is an array, dimension (LDVT, N). If JOBVT = 'A', VT contains the N-by-N unitary matrix V^H ; if JOBVT = 'S', VT contains the first min(m,n) rows of V^H (the right singular vectors, stored rowwise); if JOBVT = 'N' or 'O', VT is not referenced.

LDVT Input parameter.

LDVT is INTEGER

The leading dimension of the array VT. LDVT >= 1; if JOBVT = 'A', LDVT >= N; if JOBVT = 'S', LDVT >= min(M, N).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= MAX(1,2*MIN(M, N)+MAX(M, N)). For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (5*min(M, N)). On exit, if INFO > 0, RWORK(1:MIN(M, N)-1) contains the unconverged superdiagonal elements of an upper bidiagonal matrix B whose diagonal is in S (not necessarily sorted). B satisfies A = U * B * VT, so it has the same singular values as A, and singular vectors related by U and VT.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if CBDSQR did not converge, INFO specifies how many superdiagonals of an intermediate bidiagonal form B did not converge to zero. See the description of RWORK above for details.

Related Information

For this routine in other precisions, please see *dgesvd*, *sgesvd* and *zgesvd*. It also exists with a native C interface as *LAPACKE_cgesvd*.

4.10.7 cgesvdx

```
CGESVDX computes the singular value decomposition (SVD) of a complex
 M-by-N matrix A, optionally computing the left and/or right singular
 vectors. The SVD is written

     A = U * SIGMA * transpose(V)

 where SIGMA is an M-by-N matrix which is zero except for its
 min(m,n) diagonal elements, U is an M-by-M unitary matrix, and
 V is an N-by-N unitary matrix.  The diagonal elements of SIGMA
 are the singular values of A; they are real and non-negative, and
 are returned in descending order.  The first min(m,n) columns of
 U and V are the left and right singular vectors of A.

 CGESVDX uses an eigenvalue problem for obtaining the SVD, which
 allows for the computation of a subset of singular values and
 vectors. See SBDSVDX for details.

 Note that the routine returns V**T, not V.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine cgesvdx(JOBU, JOBVT, RANGE, M, N, A, LDA, VL, VU, IL, IU, NS, S, U,
                   LDU, VT, LDVT, WORK, LWORK, RWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgesvdx_(const char *jobu, const char *jobvt, const char *range,
              const armpl_int_t *m, const armpl_int_t *n,
              armpl_singlecomplex_t *a, const armpl_int_t *lda,
              const float *vl, const float *vu, const armpl_int_t *il,
              const armpl_int_t *iu, armpl_int_t *ns, float *s,
              armpl_singlecomplex_t *u, const armpl_int_t *ldu,
              armpl_singlecomplex_t *vt, const armpl_int_t *ldvt,
              armpl_singlecomplex_t *work, const armpl_int_t *lwork,
              float *rwork, armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

JOBU Input parameter.

JOBU is CHARACTER*1

Specifies options for computing all or part of the matrix U: = 'V': the first min(m,n) columns of U (the left singular vectors) or as specified by RANGE are returned in the array U; = 'N': no columns of U (no left singular vectors) are computed.

JOBVT Input parameter.

JOBVT is CHARACTER*1

Specifies options for computing all or part of the matrix V^T : = 'V': the first min(m,n) rows of V^T (the right singular vectors) or as specified by RANGE are returned in the array VT; = 'N': no rows of V^T (no right singular vectors) are computed.

RANGE Input parameter.

RANGE is CHARACTER*1

= ‘A’: all singular values will be found. = ‘V’: all singular values in the half-open interval (VL,VU] will be found. = ‘I’: the IL-th through IU-th singular values will be found.

M Input parameter.

M is INTEGER

The number of rows of the input matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the input matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the contents of A are destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

VL Input parameter.

VL is REAL

If RANGE=’V’, the lower bound of the interval to be searched for singular values. VU > VL. Not referenced if RANGE = ‘A’ or ‘I’.

VU Input parameter.

VU is REAL

If RANGE=’V’, the upper bound of the interval to be searched for singular values. VU > VL. Not referenced if RANGE = ‘A’ or ‘I’.

IL Input parameter.

IL is INTEGER

If RANGE=’I’, the index of the smallest singular value to be returned. 1 <= IL <= IU <= min(M, N), if min(M, N) > 0. Not referenced if RANGE = ‘A’ or ‘V’.

IU Input parameter.

IU is INTEGER

If RANGE=’I’, the index of the largest singular value to be returned. 1 <= IL <= IU <= min(M, N), if min(M, N) > 0. Not referenced if RANGE = ‘A’ or ‘V’.

NS Output parameter.

NS is INTEGER

The total number of singular values found, 0 <= NS <= min(M, N). If RANGE = ‘A’, NS = min(M, N); if RANGE = ‘I’, NS = IU-IL+1.

S Output parameter.

S is REAL

S is an array, dimension (min(M, N)). The singular values of A, sorted so that S(i) >= S(i+1).

U Output parameter.

U is COMPLEX

U is an array, dimension (LDU,UCOL). If JOBU = 'V', U contains columns of U (the left singular vectors, stored columnwise) as specified by RANGE; if JOBU = 'N', U is not referenced. Note: The user must ensure that UCOL >= NS; if RANGE = 'V', the exact value of NS is not known in advance and an upper bound must be used.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U. LDU >= 1; if JOBU = 'V', LDU >= M.

VT Output parameter.

VT is COMPLEX

VT is an array, dimension (LDVT, N). If JOBVT = 'V', VT contains the rows of V^T (the right singular vectors, stored rowwise) as specified by RANGE; if JOBVT = 'N', VT is not referenced. Note: The user must ensure that LDVT >= NS; if RANGE = 'V', the exact value of NS is not known in advance and an upper bound must be used.

LDVT Input parameter.

LDVT is INTEGER

The leading dimension of the array VT. LDVT >= 1; if JOBVT = 'V', LDVT >= NS (see above).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK;

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= MAX(1,MIN(M, N)*(MIN(M, N)+4)) for the paths (see comments inside the code): - PATH 1 (M much larger than N) - PATH 1t (N much larger than M) LWORK >= MAX(1,MIN(M, N)*2+MAX(M, N)) for the other paths. For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (MAX(1,LRWORK)). LRWORK >= MIN(M, N)*(MIN(M, N)*2+15*MIN(M, N)).

IWORK Output parameter.

IWORK is INTEGER array, dimension (12*MIN(M, N))

If INFO = 0, the first NS elements of IWORK are zero. If INFO > 0, then IWORK contains the indices of the eigenvectors that failed to converge in SBDSVDX/SSTEVX.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, then i eigenvectors failed to converge in SBDSVDX/SSTEVX. if INFO = N*2 + 1, an internal error occurred in SBDSVDX

Related Information

For this routine in other precisions, please see *dgesvdx*, *sgesvdx* and *zgesvdx*. It also exists with a native C interface as *LAPACKE_cgesvdx*.

4.10.8 cgesvj

`cgesvj` computes the singular value decomposition (SVD) of a complex M-by-N matrix A, where M >= N. The SVD of A is written as

```
                            [++]   [xx]   [x0]   [xx]
A = U * SIGMA * V^*,  [++] = [xx] * [ox] * [xx]
                            [++]   [xx]
```

where SIGMA is an N-by-N diagonal matrix, U is an M-by-N orthonormal matrix, and V is an N-by-N unitary matrix. The diagonal elements of SIGMA are the singular values of A. The columns of U and V are the left and the right singular vectors of A, respectively.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgesvj(JOBA, JOBU, JOBV, M, N, A, LDA, SVA, MV, V, LDV, CWORK,
                  LWORK, RWORK, LRWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgesvj_(const char *joba, const char *jobu, const char *jobv,
             const armpl_int_t *m, const armpl_int_t *n,
             armpl_singlecomplex_t *a, const armpl_int_t *lda, float *sva,
             const armpl_int_t *mv, armpl_singlecomplex_t *v,
             const armpl_int_t *ldv, armpl_singlecomplex_t *cwork,
             const armpl_int_t *lwork, float *rwork,
             const armpl_int_t *lrwork, armpl_int_t *info, ... );
```

Parameters

JOBA Input parameter.

JOBA is CHARACTER*1

Specifies the structure of A. = 'L': The input matrix A is lower triangular; = 'U': The input matrix A is upper triangular; = 'G': The input matrix A is general M-by-N matrix, M >= N.

JOBU Input parameter.

JOBU is CHARACTER*1

Specifies whether to compute the left singular vectors (columns of U): = 'U' or 'F': The left singular vectors corresponding to the nonzero singular values are computed and returned in the leading columns of A. See more details in the description of A. The default numerical orthogonality threshold is set to approximately TOL=CTOL*EPS, CTOL=SQRT(M), EPS=SLAMCH('E'). = 'C': Analogous to JOBU='U', except that user can control the level of numerical orthogonality of the computed left singular vectors. TOL can be set to TOL = CTOL*EPS, where CTOL is given on input in the array WORK. No CTOL smaller than ONE is allowed. CTOL greater than 1 / EPS is meaningless. The option 'C' can be used if M*EPS is satisfactory orthogonality of the computed left singular vectors, so CTOL=M could save few sweeps of Jacobi rotations.

See the descriptions of A and WORK(1). = 'N': The matrix U is not computed. However, see the description of A.

JOBV Input parameter.

JOBV is CHARACTER*1

Specifies whether to compute the right singular vectors, that is, the matrix V: = 'V' or 'J': the matrix V is computed and returned in the array V = 'A' : the Jacobi rotations are applied to the MV-by-N array V. In other words, the right singular vector matrix V is not computed explicitly; instead it is applied to an MV-by-N matrix initially stored in the first MV rows of V. = 'N' : the matrix V is not computed and the array V is not referenced

M Input parameter.

M is INTEGER

The number of rows of the input matrix A. 1/SLAMCH('E') > M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the input matrix A. M >= N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, If JOBU .EQ. 'U' .OR. JOBU .EQ. 'C': If INFO .EQ. 0 : RANKA orthonormal columns of U are returned in the leading RANKA columns of the array A. Here RANKA <= N is the number of computed singular values of A that are above the underflow threshold SLAMCH('S'). The singular vectors corresponding to underflowed or zero singular values are not computed. The value of RANKA is returned in the array RWORK as RANKA=NINT(RWORK(2)). Also see the descriptions of SVA and RWORK. The computed columns of U are mutually numerically orthogonal up to approximately TOL=SQRT(M)*EPS (default); or TOL=CTOL*EPS (JOBU.EQ.'C'), see the description of JOBU. If INFO .GT. 0, the procedure CGESVJ did not converge in the given number of iterations (sweeps). In that case, the computed columns of U may not be orthogonal up to TOL. The output U (stored in A), SIGMA (given by the computed singular values in SVA(1:N)) and V is still a decomposition of the input matrix A in the sense that the residual || A - SCALE * U * SIGMA * V^* ||_2 / ||A||_2 is small. If JOBU .EQ. 'N': If INFO .EQ. 0 : Note that the left singular vectors are 'for free' in the one-sided Jacobi SVD algorithm. However, if only the singular values are needed, the level of numerical orthogonality of U is not an issue and iterations are stopped when the columns of the iterated matrix are numerically orthogonal up to approximately M*EPS. Thus, on exit, A contains the columns of U scaled with the corresponding singular values. If INFO .GT. 0 : the procedure CGESVJ did not converge in the given number of iterations (sweeps).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

SVA Output parameter.

SVA is REAL

SVA is an array, dimension (N). On exit, If INFO .EQ. 0 : depending on the value SCALE = RWORK(1), we have: If SCALE .EQ. ONE: SVA(1:N) contains the computed singular values of A. During the computation SVA contains the Euclidean column norms of the iterated matrices in the array A. If SCALE .NE. ONE: The singular values of A are SCALE*SVA(1:N), and this factored representation is due to the fact that some of the singular values of A might underflow or overflow.

If INFO .GT. 0 : the procedure CGESVJ did not converge in the given number of iterations (sweeps) and SCALE*SVA(1:N) may not be accurate.

MV Input parameter.

MV is INTEGER

If JOBV .EQ. 'A', then the product of Jacobi rotations in CGESVJ is applied to the first MV rows of V. See the description of JOBV.

V Input and output parameter.

V is COMPLEX

V is an array, dimension (LDV, N). If JOBV = 'V', then V contains on exit the N-by-N matrix of the right singular vectors; If JOBV = 'A', then V contains the product of the computed right singular vector matrix and the initial matrix in the array V. If JOBV = 'N', then V is not referenced.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V, LDV .GE. 1. If JOBV .EQ. 'V', then LDV .GE. max(1, N). If JOBV .EQ. 'A', then LDV .GE. max(1, MV) .

CWORK Input and output parameter.

CWORK is COMPLEX

CWORK is an array, dimension (max(1, LWORK)). Used as workspace. If on entry LWORK .EQ. -1, then a workspace query is assumed and no computation is done; CWORK(1) is set to the minial (and optimal) length of CWORK.

LWORK Input parameter.

LWORK is INTEGER.

Length of CWORK, LWORK >= M+N.

RWORK Input and output parameter.

RWORK is REAL

RWORK is an array, dimension (max(6, LRWORK)). On entry, If JOBU .EQ. 'C' : RWORK(1) = CTOL, where CTOL defines the threshold for convergence. The process stops if all columns of A are mutually orthogonal up to CTOL*EPS, EPS=SLAMCH('E'). It is required that CTOL >= ONE, i.e. it is not allowed to force the routine to obtain orthogonality below EPSILON. On exit, RWORK(1) = SCALE is the scaling factor such that SCALE*SVA(1:N) are the computed singular values of A. (See description of SVA().) RWORK(2) = NINT(RWORK(2)) is the number of the computed nonzero singular values. RWORK(3) = NINT(RWORK(3)) is the number of the computed singular values that are larger than the underflow threshold. RWORK(4) = NINT(RWORK(4)) is the number of sweeps of Jacobi rotations needed for numerical convergence. RWORK(5) = max_{i.NE.j} |COS(A(:,i),A(:,j))| in the last sweep. This is useful information in cases when CGESVJ did not converge, as it can be used to estimate whether the output is stil useful and for post festum analysis. RWORK(6) = the largest absolute value over all sines of the Jacobi rotation angles in the last sweep. It can be useful for a post festum analysis. If on entry LRWORK .EQ. -1, then a workspace query is assumed and no computation is done; RWORK(1) is set to the minial (and optimal) length of RWORK.

LRWORK Input parameter.

LRWORK is INTEGER

Length of RWORK, LRWORK >= MAX(6, N).

INFO Output parameter.

INFO is INTEGER

= 0 : successful exit. < 0 : if INFO = -i, then the i-th argument had an illegal value > 0 : CGESVJ did not converge in the maximal allowed number (NSWEEP=30) of sweeps. The output may still be useful. See the description of RWORK.

Related Information

For this routine in other precisions, please see *dgesvj*, *sgesvj* and *zgesvj*. It also exists with a native C interface as *LAPACKE_cgesvj*.

4.10.9 cggsvd3

`cggsvd3` computes the generalized singular value decomposition (GSVD) of an M-by-N complex matrix A and P-by-N complex matrix B:

```
U**H*A*Q = D1*( 0 R ),    V**H*B*Q = D2*( 0 R )
```

where U, V and Q are unitary matrices. Let K+L = the effective numerical rank of the matrix $(A^H, B^H)^H$, then R is a (K+L)-by-(K+L) nonsingular upper triangular matrix, D1 and D2 are M-by-(K+L) and P-by-(K+L) "diagonal" matrices and of the following structures, respectively:

If M-K-L >= 0,

```
                  K  L
     D1 =     K ( I  0 )
              L ( 0  C )
          M-K-L ( 0  0 )

                 K  L
     D2 =   L ( 0  S )
          P-L ( 0  0 )

                N-K-L  K    L
( 0 R ) = K (  0   R11  R12 )
          L (  0    0   R22 )
```

where

```
C = diag( ALPHA(K+1), ... , ALPHA(K+L) ),
S = diag( BETA(K+1),  ... , BETA(K+L) ),
C**2 + S**2 = I.

R is stored in A(1:K+L,N-K-L+1:N) on exit.
```

If M-K-L < 0,

```
                 K M-K K+L-M
     D1 =   K ( I  0    0   )
          M-K ( 0  C    0   )

                   K M-K K+L-M
     D2 =   M-K ( 0  S    0  )
          K+L-M ( 0  0    I  )
            P-L ( 0  0    0  )

                  N-K-L  K   M-K  K+L-M
( 0 R ) =     K ( 0    R11  R12  R13  )
            M-K ( 0     0   R22  R23  )
          K+L-M ( 0     0    0   R33  )
```

where

```
C = diag( ALPHA(K+1), ... , ALPHA(M) ),
S = diag( BETA(K+1),  ... , BETA(M) ),
C**2 + S**2 = I.
```

(continues on next page)

(continued from previous page)

```
(R11 R12 R13 ) is stored in A(1:M, N-K-L+1:N), and R33 is stored
( 0  R22 R23 )
in B(M-K+1:L,N+M-K-L+1:N) on exit.
```

The routine computes C, S, R, and optionally the unitary transformation matrices U, V and Q.

In particular, if B is an N-by-N nonsingular matrix, then the GSVD of A and B implicitly gives the SVD of A*inv(B):

```
A*inv(B) = U*(D1*inv(D2))*V**H.
```

If (A^H ,B^H)^H has orthonormal columns, then the GSVD of A and B is also equal to the CS decomposition of A and B. Furthermore, the GSVD can be used to derive the solution of the eigenvalue problem:

```
A**H*A x = lambda* B**H*B x.
```

In some literature, the GSVD of A and B is presented in the form

```
U**H*A*X = ( 0 D1 ),   V**H*B*X = ( 0 D2 )
```

where U and V are orthogonal and X is nonsingular, and D1 and D2 are ‘’diagonal’‘. The former GSVD form can be converted to the latter form by taking the nonsingular matrix X as

```
X = Q*(  I   0    )
      (  0 inv(R) )
```

Syntax

Fortran specification:

```
use armpl_library

subroutine cggsvd3(JOBU, JOBV, JOBQ, M, N, P, K, L, A, LDA, B, LDB, ALPHA,
                   BETA, U, LDU, V, LDV, Q, LDQ, WORK, LWORK, RWORK, IWORK,
                   INFO)
```

C specification:

```
#include "armpl.h"

void cggsvd3_(const char *jobu, const char *jobv, const char *jobq,
              const armpl_int_t *m, const armpl_int_t *n,
              const armpl_int_t *p, armpl_int_t *k, armpl_int_t *l,
              armpl_singlecomplex_t *a, const armpl_int_t *lda,
              armpl_singlecomplex_t *b, const armpl_int_t *ldb, float *alpha,
              float *beta, armpl_singlecomplex_t *u, const armpl_int_t *ldu,
              armpl_singlecomplex_t *v, const armpl_int_t *ldv,
              armpl_singlecomplex_t *q, const armpl_int_t *ldq,
              armpl_singlecomplex_t *work, const armpl_int_t *lwork,
              float *rwork, armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

JOBU Input parameter.

JOBU is CHARACTER*1

= ‘U’: Unitary matrix U is computed; = ‘N’: U is not computed.

JOBV Input parameter.

JOBV is CHARACTER*1

= 'V': Unitary matrix V is computed; = 'N': V is not computed.

JOBQ Input parameter.

JOBQ is CHARACTER*1

= 'Q': Unitary matrix Q is computed; = 'N': Q is not computed.

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrices A and B. N >= 0.

P Input parameter.

P is INTEGER

The number of rows of the matrix B. P >= 0.

K Output parameter.

K is INTEGER

L Output parameter.

L is INTEGER

On exit, K and L specify the dimension of the subblocks described in Purpose. K + L = effective numerical rank of $(A^H, B^H)^H$.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, A contains the triangular matrix R, or part of R. See Purpose for details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, N). On entry, the P-by-N matrix B. On exit, B contains part of the triangular matrix R if M-K-L < 0. See Purpose for details.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, P).

ALPHA Output parameter.

ALPHA is REAL

ALPHA is an array, dimension (N) .

BETA Output parameter.

BETA is REAL

BETA is an array, dimension (N). On exit, ALPHA and BETA contain the generalized singular value pairs of A and B; ALPHA(1:K) = 1, BETA(1:K) = 0, and if M-K-L >= 0, ALPHA(K+1:K+L) = C, BETA(K+1:K+L) = S, or if M-K-L < 0, ALPHA(K+1:M)=C, ALPHA(M+1:K+L)=0 BETA(K+1:M) =S, BETA(M+1:K+L) =1 and ALPHA(K+L+1:N) = 0 BETA(K+L+1:N) = 0

U Output parameter.

U is COMPLEX

U is an array, dimension (LDU, M). If JOBU = 'U', U contains the M-by-M unitary matrix U. If JOBU = 'N', U is not referenced.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U. LDU >= max(1, M) if JOBU = 'U'; LDU >= 1 otherwise.

V Output parameter.

V is COMPLEX

V is an array, dimension (LDV, P). If JOBV = 'V', V contains the P-by-P unitary matrix V. If JOBV = 'N', V is not referenced.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. LDV >= max(1, P) if JOBV = 'V'; LDV >= 1 otherwise.

Q Output parameter.

Q is COMPLEX

Q is an array, dimension (LDQ, N). If JOBQ = 'Q', Q contains the N-by-N unitary matrix Q. If JOBQ = 'N', Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, N) if JOBQ = 'Q'; LDQ >= 1 otherwise.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (2*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

On exit, IWORK stores the sorting information. More precisely, the following loop will sort ALPHA for I = K+1, min(M,K+L) swap ALPHA(I) and ALPHA(IWORK(I)) endfor such that ALPHA(1) >= ALPHA(2) >= ... >= ALPHA(N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = 1, the Jacobi-type procedure failed to converge. For further details, see subroutine CTGSJA.

Related Information

For this routine in other precisions, please see *dggsvd3*, *sggsvd3* and *zggsvd3*. It also exists with a native C interface as *LAPACKE_cggsvd3*.

4.10.10 cggsvp3

cggsvp3 computes unitary matrices U, V and Q such that

```
                    N-K-L  K    L
U**H*A*Q =      K ( 0    A12  A13 )  if M-K-L >= 0;
                L ( 0     0   A23 )
            M-K-L ( 0     0    0  )

                 N-K-L  K    L
         =     K ( 0    A12  A13 )  if M-K-L < 0;
             M-K ( 0     0   A23 )

                 N-K-L  K    L
V**H*B*Q =   L ( 0     0   B13 )
             P-L ( 0     0    0  )
```

where the K-by-K matrix A12 and L-by-L matrix B13 are nonsingular upper triangular; A23 is L-by-L upper triangular if M-K-L >= 0, otherwise A23 is (M-K)-by-L upper trapezoidal. K+L = the effective numerical rank of the (M+P)-by-N matrix $(A^H, B^H)^H$.

This decomposition is the preprocessing step for computing the Generalized Singular Value Decomposition (GSVD), see subroutine CGGSVD3.

Syntax

Fortran specification:

```
use armpl_library

subroutine cggsvp3(JOBU, JOBV, JOBQ, M, P, N, A, LDA, B, LDB, TOLA, TOLB, K,
                   L, U, LDU, V, LDV, Q, LDQ, IWORK, RWORK, TAU, WORK, LWORK,
                   INFO)
```

C specification:

```
#include "armpl.h"

void cggsvp3_(const char *jobu, const char *jobv, const char *jobq,
              const armpl_int_t *m, const armpl_int_t *p,
```

(continues on next page)

(continued from previous page)

```
                const armpl_int_t *n, armpl_singlecomplex_t *a,
                const armpl_int_t *lda, armpl_singlecomplex_t *b,
                const armpl_int_t *ldb, const float *tola, const float *tolb,
                armpl_int_t *k, armpl_int_t *l, armpl_singlecomplex_t *u,
                const armpl_int_t *ldu, armpl_singlecomplex_t *v,
                const armpl_int_t *ldv, armpl_singlecomplex_t *q,
                const armpl_int_t *ldq, armpl_int_t *iwork, float *rwork,
                armpl_singlecomplex_t *tau, armpl_singlecomplex_t *work,
                const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

JOBU Input parameter.

JOBU is CHARACTER*1

= 'U': Unitary matrix U is computed; = 'N': U is not computed.

JOBV Input parameter.

JOBV is CHARACTER*1

= 'V': Unitary matrix V is computed; = 'N': V is not computed.

JOBQ Input parameter.

JOBQ is CHARACTER*1

= 'Q': Unitary matrix Q is computed; = 'N': Q is not computed.

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

P Input parameter.

P is INTEGER

The number of rows of the matrix B. P >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrices A and B. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, A contains the triangular (or trapezoidal) matrix described in the Purpose section.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, N). On entry, the P-by-N matrix B. On exit, B contains the triangular matrix described in the Purpose section.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, P).

TOLA Input parameter.

TOLA is REAL

TOLB Input parameter.

TOLB is REAL

TOLA and TOLB are the thresholds to determine the effective numerical rank of matrix B and a subblock of A. Generally, they are set to TOLA = MAX(M, N)*norm(A)*MACHEPS, TOLB = MAX(P, N)*norm(B)*MACHEPS. The size of TOLA and TOLB may affect the size of backward errors of the decomposition.

K Output parameter.

K is INTEGER

L Output parameter.

L is INTEGER

On exit, K and L specify the dimension of the subblocks described in Purpose section. K + L = effective numerical rank of $(A^H, B^H)^H$.

U Output parameter.

U is COMPLEX

U is an array, dimension (LDU, M). If JOBU = 'U', U contains the unitary matrix U. If JOBU = 'N', U is not referenced.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U. LDU >= max(1, M) if JOBU = 'U'; LDU >= 1 otherwise.

V Output parameter.

V is COMPLEX

V is an array, dimension (LDV, P). If JOBV = 'V', V contains the unitary matrix V. If JOBV = 'N', V is not referenced.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. LDV >= max(1, P) if JOBV = 'V'; LDV >= 1 otherwise.

Q Output parameter.

Q is COMPLEX

Q is an array, dimension (LDQ, N). If JOBQ = 'Q', Q contains the unitary matrix Q. If JOBQ = 'N', Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, N) if JOBQ = 'Q'; LDQ >= 1 otherwise.

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (2*N) .

TAU Output parameter.

TAU is COMPLEX

TAU is an array, dimension (N) .

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *dggsvp3*, *sggsvp3* and *zggsvp3*. It also exists with a native C interface as *LAPACKE_cggsvp3*.

4.10.11 ctgsja

ctgsja computes the generalized singular value decomposition (GSVD) of two complex upper triangular (or trapezoidal) matrices A and B.

On entry, it is assumed that matrices A and B have the following forms, which may be obtained by the preprocessing subroutine CGGSVP from a general M-by-N matrix A and P-by-N matrix B:

```
           N-K-L  K    L
A =    K ( 0    A12  A13 ) if M-K-L >= 0;
       L ( 0     0   A23 )
   M-K-L ( 0     0    0  )

         N-K-L  K    L
A =  K ( 0    A12  A13 ) if M-K-L < 0;
   M-K ( 0     0   A23 )

         N-K-L  K    L
B =  L ( 0     0   B13 )
   P-L ( 0     0    0  )
```

where the K-by-K matrix A12 and L-by-L matrix B13 are nonsingular upper triangular; A23 is L-by-L upper triangular if M-K-L >= 0, otherwise A23 is (M-K)-by-L upper trapezoidal.

On exit,

```
U**H *A*Q = D1*( 0 R ),     V**H *B*Q = D2*( 0 R ),
```

where U, V and Q are unitary matrices. R is a nonsingular upper triangular matrix, and D1 and D2 are ‘‘diagonal” matrices, which are of the following structures:

If M-K-L >= 0,

```
                    K  L
     D1 =      K ( I  0 )
               L ( 0  C )
             M-K-L ( 0  0 )

                   K  L
     D2 = L    ( 0  S )
          P-L  ( 0  0 )

                N-K-L  K    L
( 0 R ) = K (   0    R11  R12 ) K
          L (   0     0   R22 ) L
```

where

```
 C = diag( ALPHA(K+1), ... , ALPHA(K+L) ),
S = diag( BETA(K+1),  ... , BETA(K+L) ),
C**2 + S**2 = I.

R is stored in A(1:K+L,N-K-L+1:N) on exit.
```

If M-K-L < 0,

```
             K M-K K+L-M
D1 =    K ( I  0    0   )
      M-K ( 0  C    0   )

               K M-K K+L-M
D2 =    M-K ( 0  S    0   )
      K+L-M ( 0  0    I   )
        P-L ( 0  0    0   )

             N-K-L  K   M-K  K+L-M
```

(0 R) = K (0 R11 R12 R13)

```
  M-K ( 0     0    R22  R23  )
K+L-M ( 0     0     0   R33  )
```

where C = diag(ALPHA(K+1), . . . , ALPHA(M)), S = diag(BETA(K+1), . . . , BETA(M)), C**2 + S**2 = I.

R = (R11 R12 R13) is stored in A(1:M, N-K-L+1:N) and R33 is stored

```
(  0  R22 R23 )
```

in B(M-K+1:L,N+M-K-L+1:N) on exit.

The computation of the unitary transformation matrices U, V or Q is optional. These matrices may either be formed explicitly, or they may be postmultiplied into input matrices U1, V1, or Q1.

Syntax

Fortran specification:

```
use armpl_library

subroutine ctgsja(JOBU, JOBV, JOBQ, M, P, N, K, L, A, LDA, B, LDB, TOLA, TOLB,
                  ALPHA, BETA, U, LDU, V, LDV, Q, LDQ, WORK, NCYCLE, INFO)
```

C specification:

```
#include "armpl.h"

void ctgsja_(const char *jobu, const char *jobv, const char *jobq,
             const armpl_int_t *m, const armpl_int_t *p, const armpl_int_t *n,
             const armpl_int_t *k, const armpl_int_t *l,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             armpl_singlecomplex_t *b, const armpl_int_t *ldb,
             const float *tola, const float *tolb, float *alpha, float *beta,
             armpl_singlecomplex_t *u, const armpl_int_t *ldu,
             armpl_singlecomplex_t *v, const armpl_int_t *ldv,
             armpl_singlecomplex_t *q, const armpl_int_t *ldq,
             armpl_singlecomplex_t *work, armpl_int_t *ncycle,
             armpl_int_t *info, ... );
```

Parameters

JOBU Input parameter.

JOBU is CHARACTER*1

= 'U': U must contain a unitary matrix U1 on entry, and the product U1* U is returned; = 'I': U is initialized to the unit matrix, and the unitary matrix U is returned; = 'N': U is not computed.

JOBV Input parameter.

JOBV is CHARACTER*1

= 'V': V must contain a unitary matrix V1 on entry, and the product V1* V is returned; = 'I': V is initialized to the unit matrix, and the unitary matrix V is returned; = 'N': V is not computed.

JOBQ Input parameter.

JOBQ is CHARACTER*1

= 'Q': Q must contain a unitary matrix Q1 on entry, and the product Q1* Q is returned; = 'I': Q is initialized to the unit matrix, and the unitary matrix Q is returned; = 'N': Q is not computed.

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

P Input parameter.

P is INTEGER

The number of rows of the matrix B. P >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrices A and B. N >= 0.

K Input parameter.

K is INTEGER

L Input parameter.

L is INTEGER

K and L specify the subblocks in the input matrices A and B: A23 = A(K+1:MIN(K+L, M),N-L+1:N) and B13 = B(1:L,,N-L+1:N) of A and B, whose GSVD is going to be computed by CTGSJA. See Further Details.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, A(N-K+1:N,1:MIN(K+L, M)) contains the triangular matrix R or part of R. See Purpose for details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, N). On entry, the P-by-N matrix B. On exit, if necessary, B(M-K+1:L,N+M-K-L+1:N) contains a part of R. See Purpose for details.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, P).

TOLA Input parameter.

TOLA is REAL

TOLB Input parameter.

TOLB is REAL

TOLA and TOLB are the convergence criteria for the Jacobi- Kogbetliantz iteration procedure. Generally, they are the same as used in the preprocessing step, say TOLA = MAX(M, N)*norm(A)*MACHEPS, TOLB = MAX(P, N)*norm(B)*MACHEPS.

ALPHA Output parameter.

ALPHA is REAL

ALPHA is an array, dimension (N) .

BETA Output parameter.

BETA is REAL

BETA is an array, dimension (N). On exit, ALPHA and BETA contain the generalized singular value pairs of A and B; ALPHA(1:K) = 1, BETA(1:K) = 0, and if M-K-L >= 0, ALPHA(K+1:K+L) = diag(C), BETA(K+1:K+L) = diag(S), or if M-K-L < 0, ALPHA(K+1:M)= C, ALPHA(M+1:K+L)= 0 BETA(K+1:M) = S, BETA(M+1:K+L) = 1. Furthermore, if K+L < N, ALPHA(K+L+1:N) = 0 BETA(K+L+1:N) = 0.

U Input and output parameter.

U is COMPLEX

U is an array, dimension (LDU, M). On entry, if JOBU = 'U', U must contain a matrix U1 (usually the unitary matrix returned by CGGSVP). On exit, if JOBU = 'I', U contains the unitary matrix U; if JOBU = 'U', U contains the product U1*U. If JOBU = 'N', U is not referenced.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U. LDU >= max(1, M) if JOBU = 'U'; LDU >= 1 otherwise.

V Input and output parameter.

V is COMPLEX

V is an array, dimension (LDV, P). On entry, if JOBV = 'V', V must contain a matrix V1 (usually the unitary matrix returned by CGGSVP). On exit, if JOBV = 'I', V contains the unitary matrix V; if JOBV = 'V', V contains the product V1*V. If JOBV = 'N', V is not referenced.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. LDV >= max(1, P) if JOBV = 'V'; LDV >= 1 otherwise.

Q Input and output parameter.

Q is COMPLEX

Q is an array, dimension (LDQ, N). On entry, if JOBQ = 'Q', Q must contain a matrix Q1 (usually the unitary matrix returned by CGGSVP). On exit, if JOBQ = 'I', Q contains the unitary matrix Q; if JOBQ = 'Q', Q contains the product Q1*Q. If JOBQ = 'N', Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, N) if JOBQ = 'Q'; LDQ >= 1 otherwise.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

NCYCLE Output parameter.

NCYCLE is INTEGER

The number of cycles required for convergence.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. = 1: the procedure does not converge after MAXIT cycles.

Related Information

For this routine in other precisions, please see *dtgsja*, *stgsja* and *ztgsja*. It also exists with a native C interface as *LAPACKE_ctgsja*.

4.10.12 cungbr

cungbr generates one of the complex unitary matrices Q or P^H determined by CGEBRD when reducing a complex matrix A to bidiagonal form: A = Q * B * P^H . Q and P^H are defined as products of elementary reflectors H(i) or G(i) respectively.

If VECT = 'Q', A is assumed to have been an M-by-K matrix, and Q is of order M: if m >= k, Q = H(1) H(2) . . . H(k) and cungbr returns the first n columns of Q, where m >= n >= k; if m < k, Q = H(1) H(2) . . . H(m-1) and cungbr returns Q as an M-by-M matrix.

If VECT = 'P', A is assumed to have been a K-by-N matrix, and P^H is of order N: if k < n, P^H = G(k) . . . G(2) G(1) and cungbr returns the first m rows of P^H , where n >= m >= k; if k >= n, P^H = G(n-1) . . . G(2) G(1) and cungbr returns P^H as an N-by-N matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine cungbr(VECT, M, N, K, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cungbr_(const char *vect, const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *k, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, const armpl_singlecomplex_t *tau,
             armpl_singlecomplex_t *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

VECT Input parameter.

VECT is CHARACTER*1

Specifies whether the matrix Q or the matrix P^H is required, as defined in the transformation applied by CGEBRD: = 'Q': generate Q; = 'P': generate P^H .

M Input parameter.

M is INTEGER

The number of rows of the matrix Q or P^H to be returned. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix Q or P^H to be returned. N >= 0. If VECT = 'Q', M >= N >= min(M, K); if VECT = 'P', N >= M >= min(N, K).

K Input parameter.

K is INTEGER

If VECT = 'Q', the number of columns in the original M-by-K matrix reduced by CGEBRD. If VECT = 'P', the number of rows in the original K-by-N matrix reduced by CGEBRD. K >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the vectors which define the elementary reflectors, as returned by CGEBRD. On exit, the M-by-N matrix Q or P^H .

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= M.

TAU Input parameter.

TAU is COMPLEX

TAU is an array, dimension. (min(M, K)) if VECT = 'Q' (min(N, K)) if VECT = 'P' TAU(i) must contain the scalar factor of the elementary reflector H(i) or G(i), which determines Q or P^H , as returned by CGEBRD in its array argument TAUQ or TAUP.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,min(M, N)). For optimum performance LWORK >= min(M, N)*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zungbr*. It also exists with a native C interface as *LAPACKE_cungbr*.

4.10.13 cunmbr

If VECT = 'Q', cunmbr overwrites the general complex M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'C': Q^H * C C * Q^H

If VECT = 'P', cunmbr overwrites the general complex M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': P * C C * P TRANS = 'C': P^H * C C * P^H

Here Q and P^H are the unitary matrices determined by CGEBRD when reducing a complex matrix A to bidiagonal form: A = Q * B * P^H . Q and P^H are defined as products of elementary reflectors H(i) and G(i) respectively.

Let nq = m if SIDE = 'L' and nq = n if SIDE = 'R'. Thus nq is the order of the unitary matrix Q or P^H that is applied.

If VECT = 'Q', A is assumed to have been an NQ-by-K matrix: if nq >= k, Q = H(1) H(2) . . . H(k); if nq < k, Q = H(1) H(2) . . . H(nq-1).

If VECT = 'P', A is assumed to have been a K-by-NQ matrix: if k < nq, P = G(1) G(2) . . . G(k); if k >= nq, P = G(1) G(2) . . . G(nq-1).

Syntax

Fortran specification:

```
use armpl_library

subroutine cunmbr(VECT, SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK,
                  LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cunmbr_(const char *vect, const char *side, const char *trans,
             const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *k,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const armpl_singlecomplex_t *tau, armpl_singlecomplex_t *c,
             const armpl_int_t *ldc, armpl_singlecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

VECT Input parameter.

VECT is CHARACTER*1

= 'Q': apply Q or Q^H ; = 'P': apply P or P^H .

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q, Q^H , P or P^H from the Left; = 'R': apply Q, Q^H , P or P^H from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q or P; = 'C': Conjugate transpose, apply Q^H or P^H .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

If VECT = 'Q', the number of columns in the original matrix reduced by CGEBRD. If VECT = 'P', the number of rows in the original matrix reduced by CGEBRD. K >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension. (LDA,min(nq, K)) if VECT = 'Q' (LDA,nq) if VECT = 'P' The vectors which define the elementary reflectors H(i) and G(i), whose products determine the matrices Q and P, as returned by CGEBRD.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If VECT = 'Q', LDA >= max(1,nq); if VECT = 'P', LDA >= max(1,min(nq, K)).

TAU Input parameter.

TAU is COMPLEX

TAU is an array, dimension (min(nq, K)). TAU(i) must contain the scalar factor of the elementary reflector H(i) or G(i) which determines Q or P, as returned by CGEBRD in the array argument TAUQ or TAUP.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^H * C or C*Q^H or C*Q or P* C or P^H * C or C*P or C*P^H .

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, N); if SIDE = 'R', LWORK >= max(1, M); if N = 0 or M = 0, LWORK >= 1. For optimum performance LWORK >= max(1,N*NB) if SIDE = 'L', and LWORK >= max(1,M*NB) if SIDE = 'R', where NB is the optimal blocksize. (NB = 0 if M = 0 or N = 0.)

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zunmbr*. It also exists with a native C interface as *LAPACKE_cunmbr*.

4.10.14 dbdsdc

dbdsdc computes the singular value decomposition (SVD) of a real N-by-N (upper or lower) bidiagonal matrix B: B = U * S * VT, using a divide and conquer method, where S is a diagonal matrix with non-negative diagonal elements (the singular values of B), and U and VT are orthogonal matrices of left and right singular vectors, respectively. dbdsdc can be used to compute all singular values, and optionally, singular vectors or singular vectors in compact form.

This code makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none. See DLASD3 for details.

The code currently calls DLASDQ if singular values only are desired. However, it can be slightly modified to compute singular values using the divide and conquer method.

Syntax

Fortran specification:

```
use armpl_library

subroutine dbdsdc(UPLO, COMPQ, N, D, E, U, LDU, VT, LDVT, Q, IQ, WORK, IWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dbdsdc_(const char *uplo, const char *compq, const armpl_int_t *n,
             double *d, double *e, double *u, const armpl_int_t *ldu,
             double *vt, const armpl_int_t *ldvt, double *q, armpl_int_t *iq,
             double *work, armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': B is upper bidiagonal. = 'L': B is lower bidiagonal.

COMPQ Input parameter.

COMPQ is CHARACTER*1

Specifies whether singular vectors are to be computed as follows: = 'N': Compute singular values only; = 'P': Compute singular values and compute singular vectors in compact form; = 'I': Compute singular values and singular vectors.

N Input parameter.

N is INTEGER

The order of the matrix B. N >= 0.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry, the n diagonal elements of the bidiagonal matrix B. On exit, if INFO=0, the singular values of B.

E Input and output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N-1). On entry, the elements of E contain the offdiagonal elements of the bidiagonal matrix whose SVD is desired. On exit, E has been destroyed.

U Output parameter.

U is DOUBLE PRECISION

U is an array, dimension (LDU, N). If COMPQ = 'I', then: On exit, if INFO = 0, U contains the left singular vectors of the bidiagonal matrix. For other values of COMPQ, U is not referenced.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U. LDU >= 1. If singular vectors are desired, then LDU >= max(1, N).

VT Output parameter.

VT is DOUBLE PRECISION

VT is an array, dimension (LDVT, N). If COMPQ = 'I', then: On exit, if INFO = 0, VT^T contains the right singular vectors of the bidiagonal matrix. For other values of COMPQ, VT is not referenced.

LDVT Input parameter.

LDVT is INTEGER

The leading dimension of the array VT. LDVT >= 1. If singular vectors are desired, then LDVT >= max(1, N).

Q Output parameter.

Q is DOUBLE PRECISION

Q is an array, dimension (LDQ). If COMPQ = 'P', then: On exit, if INFO = 0, Q and IQ contain the left and right singular vectors in a compact form, requiring O(N log N) space instead of 2*N**2. In particular, Q contains all the DOUBLE PRECISION data in LDQ >= N*(11 + 2*SMLSIZ + 8*INT(LOG_2(N/(SMLSIZ+1)))) words of memory, where SMLSIZ is returned by ILAENV and is equal to the maximum size of the subproblems at the bottom of the computation tree (usually about 25). For other values of COMPQ, Q is not referenced.

IQ Output parameter.

IQ is INTEGER array, dimension (LDIQ)

If COMPQ = 'P', then: On exit, if INFO = 0, Q and IQ contain the left and right singular vectors in a compact form, requiring O(N log N) space instead of 2*N**2. In particular, IQ contains all INTEGER data in LDIQ >= N*(3 + 3*INT(LOG_2(N/(SMLSIZ+1)))) words of memory, where SMLSIZ is returned by ILAENV and is equal to the maximum size of the subproblems at the bottom of the computation tree (usually about 25). For other values of COMPQ, IQ is not referenced.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1,LWORK)). If COMPQ = 'N' then LWORK >= (4 * N). If COMPQ = 'P' then LWORK >= (6 * N). If COMPQ = 'I' then LWORK >= (3 * N**2 + 4 * N).

IWORK Output parameter.

IWORK is INTEGER array, dimension (8*N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: The algorithm failed to compute a singular value. The update process of divide and conquer failed.

Related Information

For this routine in other precisions, please see *sbdsdc*. It also exists with a native C interface as *LAPACKE_dbdsdc*.

4.10.15 dbdsqr

dbdsqr computes the singular values and, optionally, the right and/or left singular vectors from the singular value decomposition (SVD) of a real N-by-N (upper or lower) bidiagonal matrix B using the implicit zero-shift QR algorithm. The SVD of B has the form

```
B = Q * S * P**T
```

where S is the diagonal matrix of singular values, Q is an orthogonal matrix of left singular vectors, and P is an orthogonal matrix of right singular vectors. If left singular vectors are requested, this subroutine actually returns U*Q instead of Q, and, if right singular vectors are requested, this subroutine returns P^T *VT instead of P^T , for given real input matrices U and VT. When U and VT are the orthogonal matrices that reduce a general matrix A to bidiagonal form: A = U*B*VT, as computed by DGEBRD, then

```
A = (U*Q) * S * (P**T*VT)
```

is the SVD of A. Optionally, the subroutine may also compute Q^T *C for a given real input matrix C.

See "Computing Small Singular Values of Bidiagonal Matrices With Guaranteed High Relative Accuracy," by J. Demmel and W. Kahan, LAPACK Working Note #3 (or SIAM J. Sci. Statist. Comput. vol. 11, no. 5, pp. 873-912, Sept 1990) and "Accurate singular values and differential qd algorithms," by B. Parlett and V. Fernando, Technical Report CPAM-554, Mathematics Department, University of California at Berkeley, July 1992 for a detailed description of the algorithm.

Syntax

Fortran specification:

```
use armpl_library

subroutine dbdsqr(UPLO, N, NCVT, NRU, NCC, D, E, VT, LDVT, U, LDU, C, LDC,
                  WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dbdsqr_(const char *uplo, const armpl_int_t *n, const armpl_int_t *ncvt,
             const armpl_int_t *nru, const armpl_int_t *ncc, double *d,
             double *e, double *vt, const armpl_int_t *ldvt, double *u,
             const armpl_int_t *ldu, double *c, const armpl_int_t *ldc,
             double *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': B is upper bidiagonal; = 'L': B is lower bidiagonal.

N Input parameter.

N is INTEGER

The order of the matrix B. N >= 0.

NCVT Input parameter.

NCVT is INTEGER

The number of columns of the matrix VT. NCVT >= 0.

NRU Input parameter.

NRU is INTEGER

The number of rows of the matrix U. NRU >= 0.

NCC Input parameter.

NCC is INTEGER

The number of columns of the matrix C. NCC >= 0.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry, the n diagonal elements of the bidiagonal matrix B. On exit, if INFO=0, the singular values of B in decreasing order.

E Input and output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N-1). On entry, the N-1 offdiagonal elements of the bidiagonal matrix B. On exit, if INFO = 0, E is destroyed; if INFO > 0, D and E will contain the diagonal and superdiagonal elements of a bidiagonal matrix orthogonally equivalent to the one given as input.

VT Input and output parameter.

VT is DOUBLE PRECISION

VT is an array, dimension (LDVT, NCVT). On entry, an N-by-NCVT matrix VT. On exit, VT is overwritten by P^T * VT. Not referenced if NCVT = 0.

LDVT Input parameter.

LDVT is INTEGER

The leading dimension of the array VT. LDVT >= max(1, N) if NCVT > 0; LDVT >= 1 if NCVT = 0.

U Input and output parameter.

U is DOUBLE PRECISION

U is an array, dimension (LDU, N). On entry, an NRU-by-N matrix U. On exit, U is overwritten by U * Q. Not referenced if NRU = 0.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U. LDU >= max(1, NRU).

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, NCC). On entry, an N-by-NCC matrix C. On exit, C is overwritten by Q^T * C. Not referenced if NCC = 0.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, N) if NCC > 0; LDC >=1 if NCC = 0.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (4*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: If INFO = -i, the i-th argument had an illegal value > 0: if NCVT = NRU = NCC = 0, = 1, a split was marked by a positive value in E = 2, current block of Z not diagonalized after 30*N iterations (in inner while loop) = 3, termination criterion of outer while loop not met (program created more than N unreduced blocks) else NCVT = NRU = NCC = 0, the algorithm did not converge; D and E contain the elements of a bidiagonal matrix which is orthogonally similar to the input matrix B; if INFO = i, i elements of E have not converged to zero.

Related Information

For this routine in other precisions, please see *cbdsqr*, *sbdsqr* and *zbdsqr*. It also exists with a native C interface as *LAPACKE_dbdsqr*.

4.10.16 dbdsvdx

```
DBDSVDX computes the singular value decomposition (SVD) of a real
 N-by-N (upper or lower) bidiagonal matrix B, B = U * S * VT,
 where S is a diagonal matrix with non-negative diagonal elements
 (the singular values of B), and U and VT are orthogonal matrices
 of left and right singular vectors, respectively.

 Given an upper bidiagonal B with diagonal D = [ d_1 d_2 ... d_N ]
 and superdiagonal E = [ e_1 e_2 ... e_N-1 ], DBDSVDX computes the
 singular value decompositon of B through the eigenvalues and
 eigenvectors of the N*2-by-N*2 tridiagonal matrix

       |  0  d_1               |
       | d_1  0  e_1           |
 TGK = |     e_1  0  d_2       |
       |         d_2  .   .    |
       |              .   .   . |

 If (s,u,v) is a singular triplet of B with ||u|| = ||v|| = 1, then
 (+/-s,q), ||q|| = 1, are eigenpairs of TGK, with q = P * ( u' +/-v' ) /
 sqrt(2) = ( v_1 u_1 v_2 u_2 ... v_n u_n ) / sqrt(2), and
 P = [ e_{n+1} e_{1} e_{n+2} e_{2} ... ].

 Given a TGK matrix, one can either a) compute -s,-v and change signs
 so that the singular values (and corresponding vectors) are already in
 descending order (as in DGESVD/DGESDD) or b) compute s,v and reorder
 the values (and corresponding vectors). DBDSVDX implements a) by
 calling DSTEVX (bisection plus inverse iteration, to be replaced
 with a version of the Multiple Relative Robust Representation
 algorithm. (See P. Willems and B. Lang, A framework for the MR^3
 algorithm: theory and implementation, SIAM J. Sci. Comput.,
 35:740-766, 2013.)
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dbdsvdx(UPLO, JOBZ, RANGE, N, D, E, VL, VU, IL, IU, NS, S, Z, LDZ,
                   WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dbdsvdx_(const char *uplo, const char *jobz, const char *range,
              const armpl_int_t *n, const double *d, const double *e,
              const double *vl, const double *vu, const armpl_int_t *il,
              const armpl_int_t *iu, armpl_int_t *ns, double *s, double *z,
              const armpl_int_t *ldz, double *work, armpl_int_t *iwork,
              armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': B is upper bidiagonal; = 'L': B is lower bidiagonal.

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute singular values only; = 'V': Compute singular values and singular vectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all singular values will be found. = 'V': all singular values in the half-open interval [VL, VU) will be found. = 'I': the IL-th through IU-th singular values will be found.

N Input parameter.

N is INTEGER

The order of the bidiagonal matrix. N >= 0.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The n diagonal elements of the bidiagonal matrix B.

E Input parameter.

E is DOUBLE PRECISION

E is an array, dimension (max(1,N-1)). The (n-1) superdiagonal elements of the bidiagonal matrix B in elements 1 to N-1.

VL Input parameter.

VL is DOUBLE PRECISION

If RANGE='V', the lower bound of the interval to be searched for singular values. VU > VL. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is DOUBLE PRECISION

If RANGE='V', the upper bound of the interval to be searched for singular values. VU > VL. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest singular value to be returned. 1 <= IL <= IU <= min(M, N), if min(M, N) > 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest singular value to be returned. 1 <= IL <= IU <= min(M, N), if min(M, N) > 0. Not referenced if RANGE = 'A' or 'V'.

NS Output parameter.

NS is INTEGER

The total number of singular values found. 0 <= NS <= N. If RANGE = 'A', NS = N, and if RANGE = 'I', NS = IU-IL+1.

S Output parameter.

S is DOUBLE PRECISION

S is an array, dimension (N). The first NS elements contain the selected singular values in ascending order.

Z Output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (2*N,K)). If JOBZ = 'V', then if INFO = 0 the first NS columns of Z contain the singular vectors of the matrix B corresponding to the selected singular values, with U in rows 1 to N and V in rows N+1 to N*2, i.e. Z = [U] [V] If JOBZ = 'N', then Z is not referenced. Note: The user must ensure that at least K = NS+1 columns are supplied in the array Z; if RANGE = 'V', the exact value of NS is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(2,N*2).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (14*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (12*N)

If JOBZ = 'V', then if INFO = 0, the first NS elements of IWORK are zero. If INFO > 0, then IWORK contains the indices of the eigenvectors that failed to converge in DSTEVX.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, then i eigenvectors failed to converge in DSTEVX. The indices of the eigenvectors (as returned by DSTEVX) are stored in the array IWORK. if INFO = N*2 + 1, an internal error occurred.

Related Information

For this routine in other precisions, please see *sbdsvdx*. It also exists with a native C interface as *LAPACKE_dbdsvdx*.

4.10.17 dgbbrd

dgbbrd reduces a real general m-by-n band matrix A to upper bidiagonal form B by an orthogonal transformation: Q^T * A * P = B.

The routine computes B, and optionally forms Q or P^T , or computes Q^T *C for a given matrix C.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgbbrd(VECT, M, N, NCC, KL, KU, AB, LDAB, D, E, Q, LDQ, PT, LDPT,
                  C, LDC, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgbbrd_(const char *vect, const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *ncc, const armpl_int_t *kl,
             const armpl_int_t *ku, double *ab, const armpl_int_t *ldab,
```

(continues on next page)

(continued from previous page)

```
                double *d, double *e, double *q, const armpl_int_t *ldq,
                double *pt, const armpl_int_t *ldpt, double *c,
                const armpl_int_t *ldc, double *work, armpl_int_t *info, ... );
```

Parameters

VECT Input parameter.

VECT is CHARACTER*1

Specifies whether or not the matrices Q and P^T are to be formed. = 'N': do not form Q or P^T ; = 'Q': form Q only; = 'P': form P^T only; = 'B': form both.

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

NCC Input parameter.

NCC is INTEGER

The number of columns of the matrix C. NCC >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals of the matrix A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals of the matrix A. KU >= 0.

AB Input and output parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). On entry, the m-by-n band matrix A, stored in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(ku+1+i-j,j) = A(i,j) for max(1,j-ku)<=i<=min(m,j+kl). On exit, A is overwritten by values generated during the reduction.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array A. LDAB >= KL+KU+1.

D Output parameter.

D is DOUBLE PRECISION

D is an array, dimension (min(M, N)). The diagonal elements of the bidiagonal matrix B.

E Output parameter.

E is DOUBLE PRECISION

E is an array, dimension (min(M, N)-1). The superdiagonal elements of the bidiagonal matrix B.

Q Output parameter.

Q is DOUBLE PRECISION

Q is an array, dimension (LDQ, M). If VECT = 'Q' or 'B', the m-by-m orthogonal matrix Q. If VECT = 'N' or 'P', the array Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, M) if VECT = 'Q' or 'B'; LDQ >= 1 otherwise.

PT Output parameter.

PT is DOUBLE PRECISION

PT is an array, dimension (LDPT, N). If VECT = 'P' or 'B', the n-by-n orthogonal matrix P'. If VECT = 'N' or 'Q', the array PT is not referenced.

LDPT Input parameter.

LDPT is INTEGER

The leading dimension of the array PT. LDPT >= max(1, N) if VECT = 'P' or 'B'; LDPT >= 1 otherwise.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, NCC). On entry, an m-by-ncc matrix C. On exit, C is overwritten by Q^T *C. C is not referenced if NCC = 0.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M) if NCC > 0; LDC >= 1 if NCC = 0.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (2*max(M, N)) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cgbbrd*, *sgbbrd* and *zgbbrd*. It also exists with a native C interface as *LAPACKE_dgbbrd*.

4.10.18 dgebrd

dgebrd reduces a general real M-by-N matrix A to upper or lower bidiagonal form B by an orthogonal transformation: Q^T * A * P = B.

If m >= n, B is upper bidiagonal; if m < n, B is lower bidiagonal.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgebrd(M, N, A, LDA, D, E, TAUQ, TAUP, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgebrd_(const armpl_int_t *m, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, double *d, double *e, double *tauq,
             double *taup, double *work, const armpl_int_t *lwork,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows in the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns in the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the M-by-N general matrix to be reduced. On exit, if m >= n, the diagonal and the first superdiagonal are overwritten with the upper bidiagonal matrix B; the elements below the diagonal, with the array TAUQ, represent the orthogonal matrix Q as a product of elementary reflectors, and the elements above the first superdiagonal, with the array TAUP, represent the orthogonal matrix P as a product of elementary reflectors; if m < n, the diagonal and the first subdiagonal are overwritten with the lower bidiagonal matrix B; the elements below the first subdiagonal, with the array TAUQ, represent the orthogonal matrix Q as a product of elementary reflectors, and the elements above the diagonal, with the array TAUP, represent the orthogonal matrix P as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

D Output parameter.

D is DOUBLE PRECISION

D is an array, dimension (min(M, N)). The diagonal elements of the bidiagonal matrix B: D(i) = A(i,i).

E Output parameter.

E is DOUBLE PRECISION

E is an array, dimension (min(M, N)-1). The off-diagonal elements of the bidiagonal matrix B: if m >= n, E(i) = A(i,i+1) for i = 1,2,...,n-1; if m < n, E(i) = A(i+1,i) for i = 1,2,...,m-1.

TAUQ Output parameter.

TAUQ is DOUBLE PRECISION

TAUQ is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors which represent the orthogonal matrix Q. See Further Details.

TAUP Output parameter.

TAUP is DOUBLE PRECISION

TAUP is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors which represent the orthogonal matrix P. See Further Details.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= max(1, M, N). For optimum performance LWORK >= (M+N)*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cgebrd*, *sgebrd* and *zgebrd*. It also exists with a native C interface as *LAPACKE_dgebrd*.

4.10.19 dgejsv

dgejsv computes the singular value decomposition (SVD) of a real M-by-N matrix [A], where M >= N. The SVD of [A] is written as

```
[A] = [U] * [SIGMA] * [V]^t,
```

where [SIGMA] is an N-by-N (M-by-N) matrix which is zero except for its N diagonal elements, [U] is an M-by-N (or M-by-M) orthonormal matrix, and [V] is an N-by-N orthogonal matrix. The diagonal elements of [SIGMA] are the singular values of [A]. The columns of [U] and [V] are the left and the right singular vectors of [A], respectively. The matrices [U] and [V] are computed and stored in the arrays U and V, respectively. The diagonal of [SIGMA] is computed and stored in the array SVA. dgejsv can sometimes compute tiny singular values and their singular vectors much more accurately than other SVD routines, see below under Further Details.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgejsv(JOBA, JOBU, JOBV, JOBR, JOBT, JOBP, M, N, A, LDA, SVA, U,
                  LDU, V, LDV, WORK, LWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgejsv_(const char *joba, const char *jobu, const char *jobv,
             const char *jobr, const char *jobt, const char *jobp,
             const armpl_int_t *m, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, double *sva, double *u,
             const armpl_int_t *ldu, double *v, const armpl_int_t *ldv,
             double *work, const armpl_int_t *lwork, armpl_int_t *iwork,
             armpl_int_t *info, ... );
```

Parameters

JOBA Input parameter.

JOBA is CHARACTER*1

Specifies the level of accuracy: = 'C': This option works well (high relative accuracy) if A = B * D, with well-conditioned B and arbitrary diagonal matrix D. The accuracy cannot be spoiled by COLUMN scaling. The accuracy of the computed output depends on the condition of B, and the procedure aims at the best theoretical accuracy. The relative error max_{i=1:N}|d sigma_i| / sigma_i is bounded by f(M, N)*epsilon* cond(B), independent of D. The input matrix is preprocessed with the QRF with column pivoting. This initial preprocessing and preconditioning by a rank revealing QR factorization is common for all values of JOBA. Additional actions are specified as follows: = 'E': Computation as with 'C' with an additional estimate of the condition number of B. It provides a realistic error bound. = 'F': If A = D1 * C * D2 with ill-conditioned diagonal scalings D1, D2, and well-conditioned matrix C, this option gives higher accuracy than the 'C' option. If the structure of the input matrix is not known, and relative accuracy is desirable, then this option is advisable. The input matrix A is preprocessed with QR factorization with FULL (row and column) pivoting. = 'G' Computation as with 'F' with an additional estimate of the condition number of B, where A=D*B. If A has heavily weighted rows, then using this condition number gives too pessimistic error bound. = 'A': Small singular values are the noise and the matrix is treated as numerically rank deficient. The error in the computed singular values is bounded by f(m,n)*epsilon*||A||. The computed SVD A = U * S * V^t restores A up to f(m,n)*epsilon*||A||. This gives the procedure the licence to discard (set to zero) all singular values below N*epsilon*||A||. = 'R': Similar as in 'A'. Rank revealing property of the initial QR factorization is used do reveal (using triangular factor) a gap sigma_{r+1} < epsilon * sigma_r in which case the numerical RANK is declared to be r. The SVD is computed with absolute error bounds, but more accurately than with 'A'.

JOBU Input parameter.

JOBU is CHARACTER*1

Specifies whether to compute the columns of U: = 'U': N columns of U are returned in the array U. = 'F': full set of M left sing. vectors is returned in the array U. = 'W': U may be used as workspace of length M*N. See the description of U. = 'N': U is not computed.

JOBV Input parameter.

JOBV is CHARACTER*1

Specifies whether to compute the matrix V: = 'V': N columns of V are returned in the array V; Jacobi rotations are not explicitly accumulated. = 'J': N columns of V are returned in the array V, but they are computed as the product of Jacobi rotations. This option is allowed only if JOBU .NE. 'N', i.e. in computing the full SVD. = 'W': V may be used as workspace of length N*N. See the description of V. = 'N': V is not computed.

JOBR Input parameter.

JOBR is CHARACTER*1

Specifies the RANGE for the singular values. Issues the licence to set to zero small positive singular values if they are outside specified range. If A .NE. 0 is scaled so that the largest singular value of c* A is around DSQRT(BIG), BIG=SLAMCH('O'), then JOBR issues the licence to kill columns of A whose norm in c* A is less than DSQRT(SFMIN) (for JOBR.EQ.'R'), or less than SMALL=SFMIN/EPSLN, where SFMIN=SLAMCH('S'), EPSLN=SLAMCH('E'). = 'N': Do not kill small columns of c*A. This

option assumes that BLAS and QR factorizations and triangular solvers are implemented to work in that range. If the condition of A is greater than BIG, use DGESVJ. = 'R': RESTRICTED range for sigma(c*A) is [DSQRT(SFMIN), DSQRT(BIG)] (roughly, as described above). This option is recommended. ~~~~~~~~~~~~~~~~~~~~~~~~~~~~ For computing the singular values in the FULL range [SFMIN,BIG] use DGESVJ.

JOBT Input parameter.

JOBT is CHARACTER*1

If the matrix is square then the procedure may determine to use transposed A if A^t seems to be better with respect to convergence. If the matrix is not square, JOBT is ignored. This is subject to changes in the future. The decision is based on two values of entropy over the adjoint orbit of A^t * A. See the descriptions of WORK(6) and WORK(7). = 'T': transpose if entropy test indicates possibly faster convergence of Jacobi process if A^t is taken as input. If A is replaced with A^t, then the row pivoting is included automatically. = 'N': do not speculate. This option can be used to compute only the singular values, or the full SVD (U, SIGMA and V). For only one set of singular vectors (U or V), the caller should provide both U and V, as one of the matrices is used as workspace if the matrix A is transposed. The implementer can easily remove this constraint and make the code more complicated. See the descriptions of U and V.

JOBP Input parameter.

JOBP is CHARACTER*1

Issues the licence to introduce structured perturbations to drown denormalized numbers. This licence should be active if the denormals are poorly implemented, causing slow computation, especially in cases of fast convergence (!). For details see [1,2]. For the sake of simplicity, this perturbations are included only when the full SVD or only the singular values are requested. The implementer/user can easily add the perturbation for the cases of computing one set of singular vectors. = 'P': introduce perturbation = 'N': do not perturb

M Input parameter.

M is INTEGER

The number of rows of the input matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the input matrix A. M >= N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

SVA Output parameter.

SVA is DOUBLE PRECISION

SVA is an array, dimension (N). On exit, - For WORK(1)/WORK(2) = ONE: The singular values of A. During the computation SVA contains Euclidean column norms of the iterated matrices in the array A. - For WORK(1) .NE. WORK(2): The singular values of A are (WORK(1)/WORK(2)) * SVA(1:N). This factored form is used if sigma_max(A) overflows or if small singular values have been saved from underflow by scaling the input matrix A. - If JOBR='R' then some of the singular values may be returned as exact zeros obtained by "set to zero" because they are below the numerical rank threshold or are denormalized numbers.

U Output parameter.

U is DOUBLE PRECISION

U is an array, dimension (LDU, N). If JOBU = 'U', then U contains on exit the M-by-N matrix of the left singular vectors. If JOBU = 'F', then U contains on exit the M-by-M matrix of the left singular vectors, including an ONB of the orthogonal complement of the Range(A). If JOBU = 'W' .AND. (JOBV.EQ.'V' .AND. JOBT.EQ.'T' .AND. M.EQ.N), then U is used as workspace if the procedure replaces A with A^t. In that case, [V] is computed in U as left singular vectors of A^t and then copied back to the V array. This 'W' option is just a reminder to the caller that in this case U is reserved as workspace of length N*N. If JOBU = 'N' U is not referenced, unless JOBT='T'.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U, LDU >= 1. IF JOBU = 'U' or 'F' or 'W', then LDU >= M.

V Output parameter.

V is DOUBLE PRECISION

V is an array, dimension (LDV, N). If JOBV = 'V', 'J' then V contains on exit the N-by-N matrix of the right singular vectors; If JOBV = 'W', AND (JOBU.EQ.'U' AND JOBT.EQ.'T' AND M.EQ.N), then V is used as workspace if the pprocedure replaces A with A^t. In that case, [U] is computed in V as right singular vectors of A^t and then copied back to the U array. This 'W' option is just a reminder to the caller that in this case V is reserved as workspace of length N*N. If JOBV = 'N' V is not referenced, unless JOBT='T'.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V, LDV >= 1. If JOBV = 'V' or 'J' or 'W', then LDV >= N.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (LWORK). On exit, if N.GT.0 .AND. M.GT.0 (else not referenced), WORK(1) = SCALE = WORK(2) / WORK(1) is the scaling factor such that SCALE*SVA(1:N) are the computed singular values of A. (See the description of SVA().) WORK(2) = See the description of WORK(1). WORK(3) = SCONDA is an estimate for the condition number of column equilibrated A. (If JOBA .EQ. 'E' or 'G') SCONDA is an estimate of DSQRT(||(R^t * R)^(-1)||_1). It is computed using DPOCON. It holds N^(-1/4) * SCONDA <= ||R^(-1)||_2 <= N^(1/4) * SCONDA where R is the triangular factor from the QRF of A. However, if R is truncated and the numerical rank is determined to be strictly smaller than N, SCONDA is returned as -1, thus indicating that the smallest singular values might be lost.

If full SVD is needed, the following two condition numbers are useful for the analysis of the algorithm. They are provied for a developer/implementer who is familiar with the details of the method.

WORK(4) = an estimate of the scaled condition number of the triangular factor in the first QR factorization. WORK(5) = an estimate of the scaled condition number of the triangular factor in the second QR factorization. The following two parameters are computed if JOBT .EQ. 'T'. They are provided for a developer/implementer who is familiar with the details of the method.

WORK(6) = the entropy of A^t* A :: this is the Shannon entropy of diag(A^t*A) / Trace(A^t*A) taken as point in the probability simplex. WORK(7) = the entropy of A*A^t.

LWORK Input parameter.

LWORK is INTEGER

Length of WORK to confirm proper allocation of work space. LWORK depends on the job:

If only SIGMA is needed (JOBU.EQ.'N', JOBV.EQ.'N') and -> .. no scaled condition estimate required (JOBE.EQ.'N'): LWORK >= max(2*M+N,4*N+1,7). This is the minimal requirement. ->> For optimal performance (blocked code) the optimal value is LWORK >= max(2*M+N,3*N+(N+1)*NB,7). Here NB is the optimal block size for DGEQP3 and DGEQRF. In general, optimal LWORK is computed as LWORK >= max(2*M+N,N+LWORK(DGEQP3),N+LWORK(DGEQRF), 7). -> .. an estimate of the scaled condition number of A is required (JOBA='E', 'G'). In this case, LWORK is the maximum of the above and N*N+4*N, i.e. LWORK >= max(2*M+N,N*N+4*N,7). ->> For optimal performance (blocked code) the optimal value is LWORK >= max(2*M+N,3*N+(N+1)*NB, N*N+4*N, 7). In general, the optimal

length LWORK is computed as LWORK >= max(2*M+N,N+LWORK(DGEQP3),N+LWORK(DGEQRF), N+N*N+LWORK(DPOCON),7).

If SIGMA and the right singular vectors are needed (JOBV.EQ.'V'), -> the minimal requirement is LWORK >= max(2*M+N,4*N+1,7). -> For optimal performance, LWORK >= max(2*M+N,3*N+(N+1)*NB,7), where NB is the optimal block size for DGEQP3, DGEQRF, DGELQF, DORMLQ. In general, the optimal length LWORK is computed as LWORK >= max(2*M+N,N+LWORK(DGEQP3), N+LWORK(DPOCON), N+LWORK(DGELQF), 2*N+LWORK(DGEQRF), N+LWORK(DORMLQ)).

If SIGMA and the left singular vectors are needed -> the minimal requirement is LWORK >= max(2*M+N,4*N+1,7). -> For optimal performance: if JOBU.EQ.'U' :: LWORK >= max(2*M+N,3*N+(N+1)*NB,7), if JOBU.EQ.'F' :: LWORK >= max(2*M+N,3*N+(N+1)*NB,N+M*NB,7), where NB is the optimal block size for DGEQP3, DGEQRF, DORMQR. In general, the optimal length LWORK is computed as LWORK >= max(2*M+N,N+LWORK(DGEQP3),N+LWORK(DPOCON), 2*N+LWORK(DGEQRF), N+LWORK(DORMQR)). Here LWORK(DORMQR) equals N*NB (for JOBU.EQ.'U') or M*NB (for JOBU.EQ.'F').

If the full SVD is needed: (JOBU.EQ.'U' or JOBU.EQ.'F') and -> if JOBV.EQ.'V' the minimal requirement is LWORK >= max(2*M+N,6*N+2*N*N). -> if JOBV.EQ.'J' the minimal requirement is LWORK >= max(2*M+N, 4*N+N*N,2*N+N*N+6). -> For optimal performance, LWORK should be additionally larger than N+M*NB, where NB is the optimal block size for DORMQR.

IWORK Output parameter.

IWORK is INTEGER array, dimension (M+3*N).

On exit, IWORK(1) = the numerical rank determined after the initial QR factorization with pivoting. See the descriptions of JOBA and JOBR. IWORK(2) = the number of the computed nonzero singular values IWORK(3) = if nonzero, a warning message: If IWORK(3).EQ.1 then some of the column norms of A were denormalized floats. The requested high accuracy is not warranted by the data.

INFO Output parameter.

INFO is INTEGER

< 0 : if INFO = -i, then the i-th argument had an illegal value. = 0 : successful exit; > 0 : DGEJSV did not converge in the maximal allowed number of sweeps. The computed values may be inaccurate.

Related Information

For this routine in other precisions, please see *cgejsv*, *sgejsv* and *zgejsv*. It also exists with a native C interface as *LAPACKE_dgejsv*.

4.10.20 dgesdd

dgesdd computes the singular value decomposition (SVD) of a real M-by-N matrix A, optionally computing the left and right singular vectors. If singular vectors are desired, it uses a divide-and-conquer algorithm.

The SVD is written

```
A = U * SIGMA * transpose(V)
```

where SIGMA is an M-by-N matrix which is zero except for its min(m,n) diagonal elements, U is an M-by-M orthogonal matrix, and V is an N-by-N orthogonal matrix. The diagonal elements of SIGMA are the singular values of A; they are real and non-negative, and are returned in descending order. The first min(m,n) columns of U and V are the left and right singular vectors of A.

Note that the routine returns VT = V^T , not V.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgesdd(JOBZ, M, N, A, LDA, S, U, LDU, VT, LDVT, WORK, LWORK, IWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dgesdd_(const char *jobz, const armpl_int_t *m, const armpl_int_t *n,
             double *a, const armpl_int_t *lda, double *s, double *u,
             const armpl_int_t *ldu, double *vt, const armpl_int_t *ldvt,
             double *work, const armpl_int_t *lwork, armpl_int_t *iwork,
             armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

Specifies options for computing all or part of the matrix U: = 'A': all M columns of U and all N rows of V^T are returned in the arrays U and VT; = 'S': the first min(M, N) columns of U and the first min(M, N) rows of V^T are returned in the arrays U and VT; = 'O': If M >= N, the first N columns of U are overwritten on the array A and all rows of V^T are returned in the array VT; otherwise, all columns of U are returned in the array U and the first M rows of V^T are overwritten in the array A; = 'N': no columns of U or rows of V^T are computed.

M Input parameter.

M is INTEGER

The number of rows of the input matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the input matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, if JOBZ = 'O', A is overwritten with the first N columns of U (the left singular vectors, stored columnwise) if M >= N; A is overwritten with the first M rows of V^T (the right singular vectors, stored rowwise) otherwise. if JOBZ .ne. 'O', the contents of A are destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

S Output parameter.

S is DOUBLE PRECISION

S is an array, dimension (min(M, N)). The singular values of A, sorted so that S(i) >= S(i+1).

U Output parameter.

U is DOUBLE PRECISION

U is an array, dimension (LDU,UCOL). UCOL = M if JOBZ = 'A' or JOBZ = 'O' and M < N; UCOL = min(M, N) if JOBZ = 'S'. If JOBZ = 'A' or JOBZ = 'O' and M < N, U contains the M-by-M orthogonal matrix U; if JOBZ = 'S', U contains the first min(M, N) columns of U (the left singular vectors, stored columnwise); if JOBZ = 'O' and M >= N, or JOBZ = 'N', U is not referenced.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U. LDU >= 1; if JOBZ = 'S' or 'A' or JOBZ = 'O' and M < N, LDU >= M.

VT Output parameter.

VT is DOUBLE PRECISION

VT is an array, dimension (LDVT, N). If JOBZ = 'A' or JOBZ = 'O' and M >= N, VT contains the N-by-N orthogonal matrix V^T ; if JOBZ = 'S', VT contains the first min(M, N) rows of V^T (the right singular vectors, stored rowwise); if JOBZ = 'O' and M < N, or JOBZ = 'N', VT is not referenced.

LDVT Input parameter.

LDVT is INTEGER

The leading dimension of the array VT. LDVT >= 1; if JOBZ = 'A' or JOBZ = 'O' and M >= N, LDVT >= N; if JOBZ = 'S', LDVT >= min(M, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK;

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= 1. If LWORK = -1, a workspace query is assumed. The optimal size for the WORK array is calculated and stored in WORK(1), and no other work except argument checking is performed.

Let mx = max(M, N) and mn = min(M, N). If JOBZ = 'N', LWORK >= 3*mn + max(mx, 7*mn). If JOBZ = 'O', LWORK >= 3*mn + max(mx, 5*mn*mn + 4*mn). If JOBZ = 'S', LWORK >= 4*mn*mn + 7*mn. If JOBZ = 'A', LWORK >= 4*mn*mn + 6*mn + mx. These are not tight minimums in all cases; see comments inside code. For good performance, LWORK should generally be larger; a query is recommended.

IWORK Output parameter.

IWORK is INTEGER array, dimension (8*min(M, N))

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: DBDSDC did not converge, updating process failed.

Related Information

For this routine in other precisions, please see *cgesdd*, *sgesdd* and *zgesdd*. It also exists with a native C interface as *LAPACKE_dgesdd*.

4.10.21 dgesvd

dgesvd computes the singular value decomposition (SVD) of a real M-by-N matrix A, optionally computing the left and/or right singular vectors. The SVD is written

```
A = U * SIGMA * transpose(V)
```

where SIGMA is an M-by-N matrix which is zero except for its min(m,n) diagonal elements, U is an M-by-M orthogonal matrix, and V is an N-by-N orthogonal matrix. The diagonal elements of SIGMA are the singular values of A; they are real and non-negative, and are returned in descending order. The first min(m,n) columns of U and V are the left and right singular vectors of A.

Note that the routine returns V^T , not V.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgesvd(JOBU, JOBVT, M, N, A, LDA, S, U, LDU, VT, LDVT, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dgesvd_(const char *jobu, const char *jobvt, const armpl_int_t *m,
             const armpl_int_t *n, double *a, const armpl_int_t *lda,
             double *s, double *u, const armpl_int_t *ldu, double *vt,
             const armpl_int_t *ldvt, double *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

JOBU Input parameter.

JOBU is CHARACTER*1

Specifies options for computing all or part of the matrix U: = 'A': all M columns of U are returned in array U: = 'S': the first min(m,n) columns of U (the left singular vectors) are returned in the array U; = 'O': the first min(m,n) columns of U (the left singular vectors) are overwritten on the array A; = 'N': no columns of U (no left singular vectors) are computed.

JOBVT Input parameter.

JOBVT is CHARACTER*1

Specifies options for computing all or part of the matrix V^T : = 'A': all N rows of V^T are returned in the array VT; = 'S': the first min(m,n) rows of V^T (the right singular vectors) are returned in the array VT; = 'O': the first min(m,n) rows of V^T (the right singular vectors) are overwritten on the array A; = 'N': no rows of V^T (no right singular vectors) are computed.

JOBVT and JOBU cannot both be 'O'.

M Input parameter.

M is INTEGER

The number of rows of the input matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the input matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, if JOBU = 'O', A is overwritten with the first min(m,n) columns of U (the left singular vectors, stored columnwise); if JOBVT = 'O', A is overwritten with the first min(m,n) rows of V^T (the right singular vectors, stored rowwise); if JOBU .ne. 'O' and JOBVT .ne. 'O', the contents of A are destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

S Output parameter.

S is DOUBLE PRECISION

S is an array, dimension (min(M, N)). The singular values of A, sorted so that S(i) >= S(i+1).

U Output parameter.

U is DOUBLE PRECISION

U is an array, dimension (LDU,UCOL). (LDU, M) if JOBU = 'A' or (LDU,min(M, N)) if JOBU = 'S'. If JOBU = 'A', U contains the M-by-M orthogonal matrix U; if JOBU = 'S', U contains the first min(m,n) columns of U (the left singular vectors, stored columnwise); if JOBU = 'N' or 'O', U is not referenced.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U. LDU >= 1; if JOBU = 'S' or 'A', LDU >= M.

VT Output parameter.

VT is DOUBLE PRECISION

VT is an array, dimension (LDVT, N). If JOBVT = 'A', VT contains the N-by-N orthogonal matrix V^T ; if JOBVT = 'S', VT contains the first min(m,n) rows of V^T (the right singular vectors, stored rowwise); if JOBVT = 'N' or 'O', VT is not referenced.

LDVT Input parameter.

LDVT is INTEGER

The leading dimension of the array VT. LDVT >= 1; if JOBVT = 'A', LDVT >= N; if JOBVT = 'S', LDVT >= min(M, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK; if INFO > 0, WORK(2:MIN(M, N)) contains the unconverged superdiagonal elements of an upper bidiagonal matrix B whose diagonal is in S (not necessarily sorted). B satisfies A = U * B * VT, so it has the same singular values as A, and singular vectors related by U and VT.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= MAX(1,5*MIN(M, N)) for the paths (see comments inside code): - PATH 1 (M much larger than N, JOBU='N') - PATH 1t (N much larger than M, JOBVT='N') LWORK >= MAX(1,3*MIN(M, N) + MAX(M, N),5*MIN(M, N)) for the other paths For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if DBDSQR did not converge, INFO specifies how many superdiagonals of an intermediate bidiagonal form B did not converge to zero. See the description of WORK above for details.

Related Information

For this routine in other precisions, please see *cgesvd*, *sgesvd* and *zgesvd*. It also exists with a native C interface as *LAPACKE_dgesvd*.

4.10.22 dgesvdx

```
DGESVDX computes the singular value decomposition (SVD) of a real
 M-by-N matrix A, optionally computing the left and/or right singular
 vectors. The SVD is written

     A = U * SIGMA * transpose(V)

 where SIGMA is an M-by-N matrix which is zero except for its
 min(m,n) diagonal elements, U is an M-by-M orthogonal matrix, and
 V is an N-by-N orthogonal matrix.  The diagonal elements of SIGMA
 are the singular values of A; they are real and non-negative, and
 are returned in descending order.  The first min(m,n) columns of
 U and V are the left and right singular vectors of A.

 DGESVDX uses an eigenvalue problem for obtaining the SVD, which
 allows for the computation of a subset of singular values and
 vectors. See DBDSVDX for details.

 Note that the routine returns V**T, not V.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dgesvdx(JOBU, JOBVT, RANGE, M, N, A, LDA, VL, VU, IL, IU, NS, S, U,
                   LDU, VT, LDVT, WORK, LWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgesvdx_(const char *jobu, const char *jobvt, const char *range,
              const armpl_int_t *m, const armpl_int_t *n, double *a,
              const armpl_int_t *lda, const double *vl, const double *vu,
              const armpl_int_t *il, const armpl_int_t *iu, armpl_int_t *ns,
              double *s, double *u, const armpl_int_t *ldu, double *vt,
              const armpl_int_t *ldvt, double *work, const armpl_int_t *lwork,
              armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

JOBU Input parameter.

JOBU is CHARACTER*1

Specifies options for computing all or part of the matrix U: = 'V': the first min(m,n) columns of U (the left singular vectors) or as specified by RANGE are returned in the array U; = 'N': no columns of U (no left singular vectors) are computed.

JOBVT Input parameter.

JOBVT is CHARACTER*1

Specifies options for computing all or part of the matrix V^T : = 'V': the first min(m,n) rows of V^T (the right singular vectors) or as specified by RANGE are returned in the array VT; = 'N': no rows of V^T (no right singular vectors) are computed.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all singular values will be found. = 'V': all singular values in the half-open interval (VL,VU] will be found. = 'I': the IL-th through IU-th singular values will be found.

M Input parameter.

M is INTEGER

The number of rows of the input matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the input matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the contents of A are destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

VL Input parameter.

VL is DOUBLE PRECISION

If RANGE='V', the lower bound of the interval to be searched for singular values. VU > VL. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is DOUBLE PRECISION

If RANGE='V', the upper bound of the interval to be searched for singular values. VU > VL. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest singular value to be returned. 1 <= IL <= IU <= min(M, N), if min(M, N) > 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest singular value to be returned. 1 <= IL <= IU <= min(M, N), if min(M, N) > 0. Not referenced if RANGE = 'A' or 'V'.

NS Output parameter.

NS is INTEGER

The total number of singular values found, 0 <= NS <= min(M, N). If RANGE = 'A', NS = min(M, N); if RANGE = 'I', NS = IU-IL+1.

S Output parameter.

S is DOUBLE PRECISION

S is an array, dimension (min(M, N)). The singular values of A, sorted so that S(i) >= S(i+1).

U Output parameter.

U is DOUBLE PRECISION

U is an array, dimension (LDU,UCOL). If JOBU = 'V', U contains columns of U (the left singular vectors, stored columnwise) as specified by RANGE; if JOBU = 'N', U is not referenced. Note: The user must ensure that UCOL >= NS; if RANGE = 'V', the exact value of NS is not known in advance and an upper bound must be used.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U. LDU >= 1; if JOBU = 'V', LDU >= M.

VT Output parameter.

VT is DOUBLE PRECISION

VT is an array, dimension (LDVT, N). If JOBVT = 'V', VT contains the rows of V^T (the right singular vectors, stored rowwise) as specified by RANGE; if JOBVT = 'N', VT is not referenced. Note: The user must ensure that LDVT >= NS; if RANGE = 'V', the exact value of NS is not known in advance and an upper bound must be used.

LDVT Input parameter.

LDVT is INTEGER

The leading dimension of the array VT. LDVT >= 1; if JOBVT = 'V', LDVT >= NS (see above).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK;

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= MAX(1,MIN(M, N)*(MIN(M, N)+4)) for the paths (see comments inside the code): - PATH 1 (M much larger than N) - PATH 1t (N much larger than M) LWORK >= MAX(1,MIN(M, N)*2+MAX(M, N)) for the other paths. For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (12*MIN(M, N))

If INFO = 0, the first NS elements of IWORK are zero. If INFO > 0, then IWORK contains the indices of the eigenvectors that failed to converge in DBDSVDX/DSTEVX.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, then i eigenvectors failed to converge in DBDSVDX/DSTEVX. if INFO = N*2 + 1, an internal error occurred in DBDSVDX

Related Information

For this routine in other precisions, please see *cgesvdx*, *sgesvdx* and *zgesvdx*. It also exists with a native C interface as *LAPACKE_dgesvdx*.

4.10.23 dgesvj

dgesvj computes the singular value decomposition (SVD) of a real M-by-N matrix A, where M >= N. The SVD of A is written as

```
                          [++]   [xx]   [x0]   [xx]
A = U * SIGMA * V^t,      [++] = [xx] * [ox] * [xx]
                          [++]   [xx]
```

where SIGMA is an N-by-N diagonal matrix, U is an M-by-N orthonormal matrix, and V is an N-by-N orthogonal matrix. The diagonal elements of SIGMA are the singular values of A. The columns of U and V are the left and the right singular vectors of A, respectively. dgesvj can sometimes compute tiny singular values and their singular vectors much more accurately than other SVD routines, see below under Further Details.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgesvj(JOBA, JOBU, JOBV, M, N, A, LDA, SVA, MV, V, LDV, WORK,
                  LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgesvj_(const char *joba, const char *jobu, const char *jobv,
             const armpl_int_t *m, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, double *sva, const armpl_int_t *mv,
             double *v, const armpl_int_t *ldv, double *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

JOBA Input parameter.

JOBA is CHARACTER*1

Specifies the structure of A. = 'L': The input matrix A is lower triangular; = 'U': The input matrix A is upper triangular; = 'G': The input matrix A is general M-by-N matrix, M >= N.

JOBU Input parameter.

JOBU is CHARACTER*1

Specifies whether to compute the left singular vectors (columns of U): = 'U': The left singular vectors corresponding to the nonzero singular values are computed and returned in the leading columns of A. See more details in the description of A. The default numerical orthogonality threshold is set to approximately TOL=CTOL*EPS, CTOL=DSQRT(M), EPS=DLAMCH('E'). = 'C': Analogous to JOBU='U', except that user can control the level of numerical orthogonality of the computed left singular vectors. TOL can be set to TOL = CTOL*EPS, where CTOL is given on input in the array WORK. No CTOL smaller than ONE is allowed. CTOL greater than 1 / EPS is meaningless. The option 'C' can be used if M*EPS is satisfactory orthogonality of the computed left singular vectors, so CTOL=M could save few sweeps of Jacobi rotations. See the descriptions of A and WORK(1). = 'N': The matrix U is not computed. However, see the description of A.

JOBV Input parameter.

JOBV is CHARACTER*1

Specifies whether to compute the right singular vectors, that is, the matrix V: = 'V' : the matrix V is computed and returned in the array V = 'A' : the Jacobi rotations are applied to the MV-by-N array V. In other words, the right singular vector matrix V is not computed explicitly, instead it is applied to an MV-by-N matrix initially stored in the first MV rows of V. = 'N' : the matrix V is not computed and the array V is not referenced

M Input parameter.

M is INTEGER

The number of rows of the input matrix A. 1/DLAMCH('E') > M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the input matrix A. M >= N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit : If JOBU .EQ. 'U' .OR. JOBU .EQ. 'C' : If INFO .EQ. 0 : RANKA orthonormal columns of U are returned in the leading RANKA columns of the array A. Here RANKA <= N is the number of computed singular values of A that are above the underflow threshold DLAMCH('S'). The singular vectors corresponding to underflowed or zero singular values are not computed. The value of RANKA is returned in the array WORK as RANKA=NINT(WORK(2)). Also see the descriptions of SVA and WORK. The computed columns of U are mutually numerically orthogonal up to approximately TOL=DSQRT(M)*EPS (default); or TOL=CTOL*EPS (JOBU.EQ.'C'), see the description of JOBU. If INFO .GT. 0 : the procedure DGESVJ did not converge in the given number of iterations (sweeps). In that case, the computed columns of U may not be orthogonal up to TOL. The output U (stored in A), SIGMA (given by the computed singular values in SVA(1:N)) and V is still a decomposition of the input matrix A in the sense that the residual ||A-SCALE*U*SIGMA*V^T||_2 / ||A||_2 is small.

If JOBU .EQ. 'N' : If INFO .EQ. 0 : Note that the left singular vectors are 'for free' in the one-sided Jacobi SVD algorithm. However, if only the singular values are needed, the level of numerical orthogonality of U is not an issue and iterations are stopped when the columns of the iterated matrix are numerically orthogonal up to approximately M*EPS. Thus, on exit, A contains the columns of U scaled with the corresponding singular values. If INFO .GT. 0 : the procedure DGESVJ did not converge in the given number of iterations (sweeps).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

SVA Output parameter.

SVA is DOUBLE PRECISION

SVA is an array, dimension (N). On exit : If INFO .EQ. 0 : depending on the value SCALE = WORK(1), we have: If SCALE .EQ. ONE : SVA(1:N) contains the computed singular values of A. During the computation SVA contains the Euclidean column norms of the iterated matrices in the array A. If SCALE .NE. ONE : The singular values of A are SCALE*SVA(1:N), and this factored representation is due to the fact that some

of the singular values of A might underflow or overflow. If INFO .GT. 0 : the procedure DGESVJ did not converge in the given number of iterations (sweeps) and SCALE*SVA(1:N) may not be accurate.

MV Input parameter.

MV is INTEGER

If JOBV .EQ. 'A', then the product of Jacobi rotations in DGESVJ is applied to the first MV rows of V. See the description of JOBV.

V Input and output parameter.

V is DOUBLE PRECISION

V is an array, dimension (LDV, N). If JOBV = 'V', then V contains on exit the N-by-N matrix of the right singular vectors; If JOBV = 'A', then V contains the product of the computed right singular vector matrix and the initial matrix in the array V. If JOBV = 'N', then V is not referenced.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V, LDV .GE. 1. If JOBV .EQ. 'V', then LDV .GE. max(1, N). If JOBV .EQ. 'A', then LDV .GE. max(1, MV) .

WORK Input and output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (LWORK). On entry : If JOBU .EQ. 'C' : WORK(1) = CTOL, where CTOL defines the threshold for convergence. The process stops if all columns of A are mutually orthogonal up to CTOL*EPS, EPS=DLAMCH('E'). It is required that CTOL >= ONE, i.e. it is not allowed to force the routine to obtain orthogonality below EPS. On exit : WORK(1) = SCALE is the scaling factor such that SCALE*SVA(1:N) are the computed singular values of A. (See description of SVA().) WORK(2) = NINT(WORK(2)) is the number of the computed nonzero singular values. WORK(3) = NINT(WORK(3)) is the number of the computed singular values that are larger than the underflow threshold. WORK(4) = NINT(WORK(4)) is the number of sweeps of Jacobi rotations needed for numerical convergence. WORK(5) = max_{i.NE.j} |COS(A(:,i),A(:,j))| in the last sweep. This is useful information in cases when DGESVJ did not converge, as it can be used to estimate whether the output is stil useful and for post festum analysis. WORK(6) = the largest absolute value over all sines of the Jacobi rotation angles in the last sweep. It can be useful for a post festum analysis.

LWORK Input parameter.

LWORK is INTEGER

length of WORK, WORK >= MAX(6,M+N)

INFO Output parameter.

INFO is INTEGER

= 0 : successful exit. < 0 : if INFO = -i, then the i-th argument had an illegal value > 0 : DGESVJ did not converge in the maximal allowed number (30) of sweeps. The output may still be useful. See the description of WORK.

Related Information

For this routine in other precisions, please see *cgesvj*, *sgesvj* and *zgesvj*. It also exists with a native C interface as *LAPACKE_dgesvj*.

4.10.24 dggsvd3

dggsvd3 computes the generalized singular value decomposition (GSVD) of an M-by-N real matrix A and P-by-N real matrix B:

```
U**T*A*Q = D1*( 0 R ),     V**T*B*Q = D2*( 0 R )
```

where U, V and Q are orthogonal matrices. Let K+L = the effective numerical rank of the matrix $(A^T, B^T)^T$, then R is a K+L-by-K+L nonsingular upper triangular matrix, D1 and D2 are M-by-(K+L) and P-by-(K+L) "diagonal" matrices and of the following structures, respectively:

If M-K-L >= 0,

```
                   K  L
     D1 =      K ( I  0 )
               L ( 0  C )
           M-K-L ( 0  0 )

                 K  L
     D2 =    L ( 0  S )
           P-L ( 0  0 )

                N-K-L  K    L
( 0 R ) = K (   0    R11  R12 )
          L (   0     0   R22 )
```

where

```
 C = diag( ALPHA(K+1), ... , ALPHA(K+L) ),
S = diag( BETA(K+1),  ... , BETA(K+L) ),
C**2 + S**2 = I.

R is stored in A(1:K+L,N-K-L+1:N) on exit.
```

If M-K-L < 0,

```
                 K M-K K+L-M
     D1 =    K ( I  0    0   )
           M-K ( 0  C    0   )

                   K M-K K+L-M
     D2 =    M-K ( 0  S    0  )
           K+L-M ( 0  0    I  )
             P-L ( 0  0    0  )

                  N-K-L  K   M-K  K+L-M
( 0 R ) =      K ( 0    R11  R12  R13  )
             M-K ( 0      0   R22  R23  )
           K+L-M ( 0      0    0   R33  )
```

where

```
C = diag( ALPHA(K+1), ... , ALPHA(M) ),
S = diag( BETA(K+1),  ... , BETA(M) ),
C**2 + S**2 = I.

(R11 R12 R13 ) is stored in A(1:M, N-K-L+1:N), and R33 is stored
( 0  R22 R23 )
in B(M-K+1:L,N+M-K-L+1:N) on exit.
```

The routine computes C, S, R, and optionally the orthogonal transformation matrices U, V and Q.

In particular, if B is an N-by-N nonsingular matrix, then the GSVD of A and B implicitly gives the SVD of A*inv(B):

```
A*inv(B) = U*(D1*inv(D2))*V**T.
```

If (A^T ,B^T)^T has orthonormal columns, then the GSVD of A and B is also equal to the CS decomposition of A and B. Furthermore, the GSVD can be used to derive the solution of the eigenvalue problem:

```
A**T*A x = lambda* B**T*B x.
```

In some literature, the GSVD of A and B is presented in the form

```
U**T*A*X = ( 0 D1 ),   V**T*B*X = ( 0 D2 )
```

where U and V are orthogonal and X is nonsingular, D1 and D2 are ‘’diagonal’‘. The former GSVD form can be converted to the latter form by taking the nonsingular matrix X as

```
X = Q*( I   0    )
      ( 0 inv(R) ).
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dggsvd3(JOBU, JOBV, JOBQ, M, N, P, K, L, A, LDA, B, LDB, ALPHA,
                   BETA, U, LDU, V, LDV, Q, LDQ, WORK, LWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dggsvd3_(const char *jobu, const char *jobv, const char *jobq,
              const armpl_int_t *m, const armpl_int_t *n,
              const armpl_int_t *p, armpl_int_t *k, armpl_int_t *l, double *a,
              const armpl_int_t *lda, double *b, const armpl_int_t *ldb,
              double *alpha, double *beta, double *u, const armpl_int_t *ldu,
              double *v, const armpl_int_t *ldv, double *q,
              const armpl_int_t *ldq, double *work, const armpl_int_t *lwork,
              armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

JOBU Input parameter.

JOBU is CHARACTER*1

= ‘U’: Orthogonal matrix U is computed; = ‘N’: U is not computed.

JOBV Input parameter.

JOBV is CHARACTER*1

= ‘V’: Orthogonal matrix V is computed; = ‘N’: V is not computed.

JOBQ Input parameter.

JOBQ is CHARACTER*1

= ‘Q’: Orthogonal matrix Q is computed; = ‘N’: Q is not computed.

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrices A and B. N >= 0.

P Input parameter.

P is INTEGER

The number of rows of the matrix B. P >= 0.

K Output parameter.

K is INTEGER

L Output parameter.

L is INTEGER

On exit, K and L specify the dimension of the subblocks described in Purpose. K + L = effective numerical rank of $(A^T, B^T)^T$.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, A contains the triangular matrix R, or part of R. See Purpose for details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). On entry, the P-by-N matrix B. On exit, B contains the triangular matrix R if M-K-L < 0. See Purpose for details.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, P).

ALPHA Output parameter.

ALPHA is DOUBLE PRECISION

ALPHA is an array, dimension (N) .

BETA Output parameter.

BETA is DOUBLE PRECISION

BETA is an array, dimension (N). On exit, ALPHA and BETA contain the generalized singular value pairs of A and B; ALPHA(1:K) = 1, BETA(1:K) = 0, and if M-K-L >= 0, ALPHA(K+1:K+L) = C, BETA(K+1:K+L) = S, or if M-K-L < 0, ALPHA(K+1:M)=C, ALPHA(M+1:K+L)=0 BETA(K+1:M) =S, BETA(M+1:K+L) =1 and ALPHA(K+L+1:N) = 0 BETA(K+L+1:N) = 0

U Output parameter.

U is DOUBLE PRECISION

U is an array, dimension (LDU, M). If JOBU = 'U', U contains the M-by-M orthogonal matrix U. If JOBU = 'N', U is not referenced.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U. LDU >= max(1, M) if JOBU = 'U'; LDU >= 1 otherwise.

V Output parameter.

V is DOUBLE PRECISION

V is an array, dimension (LDV, P). If JOBV = 'V', V contains the P-by-P orthogonal matrix V. If JOBV = 'N', V is not referenced.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. LDV >= max(1, P) if JOBV = 'V'; LDV >= 1 otherwise.

Q Output parameter.

Q is DOUBLE PRECISION

Q is an array, dimension (LDQ, N). If JOBQ = 'Q', Q contains the N-by-N orthogonal matrix Q. If JOBQ = 'N', Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, N) if JOBQ = 'Q'; LDQ >= 1 otherwise.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

On exit, IWORK stores the sorting information. More precisely, the following loop will sort ALPHA for I = K+1, min(M,K+L) swap ALPHA(I) and ALPHA(IWORK(I)) endfor such that ALPHA(1) >= ALPHA(2) >= ... >= ALPHA(N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = 1, the Jacobi-type procedure failed to converge. For further details, see subroutine DTGSJA.

Related Information

For this routine in other precisions, please see *cggsvd3*, *sggsvd3* and *zggsvd3*. It also exists with a native C interface as *LAPACKE_dggsvd3*.

4.10.25 dggsvp3

dggsvp3 computes orthogonal matrices U, V and Q such that

```
                   N-K-L  K    L
U**T*A*Q =      K ( 0    A12  A13 )  if M-K-L >= 0;
                L ( 0     0   A23 )
            M-K-L ( 0     0    0  )

                 N-K-L  K    L
        =     K ( 0    A12  A13 )  if M-K-L < 0;
            M-K ( 0     0   A23 )

                 N-K-L  K    L
V**T*B*Q =   L ( 0     0   B13 )
            P-L ( 0     0    0  )
```

where the K-by-K matrix A12 and L-by-L matrix B13 are nonsingular upper triangular; A23 is L-by-L upper triangular if M-K-L >= 0, otherwise A23 is (M-K)-by-L upper trapezoidal. K+L = the effective numerical rank of the (M+P)-by-N matrix $(A^T, B^T)^T$.

This decomposition is the preprocessing step for computing the Generalized Singular Value Decomposition (GSVD), see subroutine DGGSVD3.

Syntax

Fortran specification:

```
use armpl_library

subroutine dggsvp3(JOBU, JOBV, JOBQ, M, P, N, A, LDA, B, LDB, TOLA, TOLB, K,
                   L, U, LDU, V, LDV, Q, LDQ, IWORK, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dggsvp3_(const char *jobu, const char *jobv, const char *jobq,
              const armpl_int_t *m, const armpl_int_t *p,
              const armpl_int_t *n, double *a, const armpl_int_t *lda,
              double *b, const armpl_int_t *ldb, const double *tola,
              const double *tolb, armpl_int_t *k, armpl_int_t *l, double *u,
              const armpl_int_t *ldu, double *v, const armpl_int_t *ldv,
              double *q, const armpl_int_t *ldq, armpl_int_t *iwork,
              double *tau, double *work, const armpl_int_t *lwork,
              armpl_int_t *info, ... );
```

Parameters

JOBU Input parameter.

JOBU is CHARACTER*1

= 'U': Orthogonal matrix U is computed; = 'N': U is not computed.

JOBV Input parameter.

JOBV is CHARACTER*1

= 'V': Orthogonal matrix V is computed; = 'N': V is not computed.

JOBQ Input parameter.

JOBQ is CHARACTER*1

= 'Q': Orthogonal matrix Q is computed; = 'N': Q is not computed.

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

P Input parameter.

P is INTEGER

The number of rows of the matrix B. P >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrices A and B. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, A contains the triangular (or trapezoidal) matrix described in the Purpose section.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). On entry, the P-by-N matrix B. On exit, B contains the triangular matrix described in the Purpose section.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, P).

TOLA Input parameter.

TOLA is DOUBLE PRECISION

TOLB Input parameter.

TOLB is DOUBLE PRECISION

TOLA and TOLB are the thresholds to determine the effective numerical rank of matrix B and a subblock of A. Generally, they are set to TOLA = MAX(M, N)*norm(A)*MACHEPS, TOLB = MAX(P, N)*norm(B)*MACHEPS. The size of TOLA and TOLB may affect the size of backward errors of the decomposition.

K Output parameter.

K is INTEGER

L Output parameter.

L is INTEGER

On exit, K and L specify the dimension of the subblocks described in Purpose section. K + L = effective numerical rank of $(A^T, B^T)^T$.

U Output parameter.

U is DOUBLE PRECISION

U is an array, dimension (LDU, M). If JOBU = 'U', U contains the orthogonal matrix U. If JOBU = 'N', U is not referenced.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U. LDU >= max(1, M) if JOBU = 'U'; LDU >= 1 otherwise.

V Output parameter.

V is DOUBLE PRECISION

V is an array, dimension (LDV, P). If JOBV = 'V', V contains the orthogonal matrix V. If JOBV = 'N', V is not referenced.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. LDV >= max(1, P) if JOBV = 'V'; LDV >= 1 otherwise.

Q Output parameter.

Q is DOUBLE PRECISION

Q is an array, dimension (LDQ, N). If JOBQ = 'Q', Q contains the orthogonal matrix Q. If JOBQ = 'N', Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, N) if JOBQ = 'Q'; LDQ >= 1 otherwise.

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

TAU Output parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (N) .

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cggsvp3*, *sggsvp3* and *zggsvp3*. It also exists with a native C interface as *LAPACKE_dggsvp3*.

4.10.26 dorgbr

`dorgbr` generates one of the real orthogonal matrices Q or P^T determined by DGEBRD when reducing a real matrix A to bidiagonal form: A = Q * B * P^T . Q and P^T are defined as products of elementary reflectors H(i) or G(i) respectively.

If VECT = 'Q', A is assumed to have been an M-by-K matrix, and Q is of order M: if m >= k, Q = H(1) H(2) . . . H(k) and `dorgbr` returns the first n columns of Q, where m >= n >= k; if m < k, Q = H(1) H(2) . . . H(m-1) and `dorgbr` returns Q as an M-by-M matrix.

If VECT = 'P', A is assumed to have been a K-by-N matrix, and P^T is of order N: if k < n, P^T = G(k) . . . G(2) G(1) and `dorgbr` returns the first m rows of P^T , where n >= m >= k; if k >= n, P^T = G(n-1) . . . G(2) G(1) and `dorgbr` returns P^T as an N-by-N matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine dorgbr(VECT, M, N, K, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dorgbr_(const char *vect, const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *k, double *a, const armpl_int_t *lda,
             const double *tau, double *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

VECT Input parameter.

`VECT` is `CHARACTER*1`

Specifies whether the matrix Q or the matrix P^T is required, as defined in the transformation applied by DGEBRD: = 'Q': generate Q; = 'P': generate P^T .

M Input parameter.

`M` is `INTEGER`

The number of rows of the matrix Q or P^T to be returned. `M` >= 0.

N Input parameter.

`N` is `INTEGER`

The number of columns of the matrix Q or P^T to be returned. `N` >= 0. If `VECT` = 'Q', `M` >= `N` >= min(`M`, `K`); if `VECT` = 'P', `N` >= `M` >= min(`N`, `K`).

K Input parameter.

`K` is `INTEGER`

If VECT = 'Q', the number of columns in the original M-by-K matrix reduced by DGEBRD. If VECT = 'P', the number of rows in the original K-by-N matrix reduced by DGEBRD. K >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the vectors which define the elementary reflectors, as returned by DGEBRD. On exit, the M-by-N matrix Q or P^T .

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Input parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension. (min(M, K)) if VECT = 'Q' (min(N, K)) if VECT = 'P' TAU(i) must contain the scalar factor of the elementary reflector H(i) or G(i), which determines Q or P^T , as returned by DGEBRD in its array argument TAUQ or TAUP.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,min(M, N)). For optimum performance LWORK >= min(M, N)*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *sorgbr*. It also exists with a native C interface as *LAPACKE_dorgbr*.

4.10.27 dormbr

If VECT = 'Q', dormbr overwrites the general real M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'T': Q^T * C C * Q^T

If VECT = 'P', dormbr overwrites the general real M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': P * C C * P TRANS = 'T': P^T * C C * P^T

Here Q and P^T are the orthogonal matrices determined by DGEBRD when reducing a real matrix A to bidiagonal form: A = Q * B * P^T . Q and P^T are defined as products of elementary reflectors H(i) and G(i) respectively.

Let nq = m if SIDE = 'L' and nq = n if SIDE = 'R'. Thus nq is the order of the orthogonal matrix Q or P^T that is applied.

If VECT = 'Q', A is assumed to have been an NQ-by-K matrix: if nq >= k, Q = H(1) H(2) . . . H(k); if nq < k, Q = H(1) H(2) . . . H(nq-1).

If VECT = 'P', A is assumed to have been a K-by-NQ matrix: if k < nq, P = G(1) G(2) . . . G(k); if k >= nq, P = G(1) G(2) . . . G(nq-1).

Syntax

Fortran specification:

```
use armpl_library

subroutine dormbr(VECT, SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK,
                  LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dormbr_(const char *vect, const char *side, const char *trans,
             const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *k,
             const double *a, const armpl_int_t *lda, const double *tau,
             double *c, const armpl_int_t *ldc, double *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

VECT Input parameter.

VECT is CHARACTER*1

= 'Q': apply Q or Q^T ; = 'P': apply P or P^T .

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q, Q^T , P or P^T from the Left; = 'R': apply Q, Q^T , P or P^T from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q or P; = 'T': Transpose, apply Q^T or P^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

If VECT = 'Q', the number of columns in the original matrix reduced by DGEBRD. If VECT = 'P', the number of rows in the original matrix reduced by DGEBRD. K >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension. (LDA,min(nq, K)) if VECT = 'Q' (LDA,nq) if VECT = 'P' The vectors which define the elementary reflectors H(i) and G(i), whose products determine the matrices Q and P, as returned by DGEBRD.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If VECT = 'Q', LDA >= max(1,nq); if VECT = 'P', LDA >= max(1,min(nq, K)).

TAU Input parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (min(nq, K)). TAU(i) must contain the scalar factor of the elementary reflector H(i) or G(i) which determines Q or P, as returned by DGEBRD in the array argument TAUQ or TAUP.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q or P* C or P^T * C or C*P or C*P^T .

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, N); if SIDE = 'R', LWORK >= max(1, M). For optimum performance LWORK >= N*NB if SIDE = 'L', and LWORK >= M*NB if SIDE = 'R', where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *sormbr*. It also exists with a native C interface as *LAPACKE_dormbr*.

4.10.28 dtgsja

dtgsja computes the generalized singular value decomposition (GSVD) of two real upper triangular (or trapezoidal) matrices A and B.

On entry, it is assumed that matrices A and B have the following forms, which may be obtained by the preprocessing subroutine DGGSVP from a general M-by-N matrix A and P-by-N matrix B:

```
           N-K-L  K    L
A =    K ( 0    A12  A13 ) if M-K-L >= 0;
       L ( 0     0   A23 )
   M-K-L ( 0     0    0  )

         N-K-L  K    L
A =  K ( 0    A12  A13 ) if M-K-L < 0;
   M-K ( 0     0   A23 )

         N-K-L  K    L
B =  L ( 0     0   B13 )
   P-L ( 0     0    0  )
```

where the K-by-K matrix A12 and L-by-L matrix B13 are nonsingular upper triangular; A23 is L-by-L upper triangular if M-K-L >= 0, otherwise A23 is (M-K)-by-L upper trapezoidal.

On exit,

```
U**T *A*Q = D1*( 0 R ),    V**T *B*Q = D2*( 0 R ),
```

where U, V and Q are orthogonal matrices. R is a nonsingular upper triangular matrix, and D1 and D2 are ‘’diagonal” matrices, which are of the following structures:

If M-K-L >= 0,

```
                 K  L
     D1 =     K ( I  0 )
              L ( 0  C )
            M-K-L ( 0  0 )

                 K  L
     D2 = L    ( 0  S )
            P-L ( 0  0 )

               N-K-L  K    L
( 0 R ) = K (  0   R11  R12 ) K
          L (  0    0   R22 ) L
```

where

```
C = diag( ALPHA(K+1), ... , ALPHA(K+L) ),
S = diag( BETA(K+1),  ... , BETA(K+L) ),
C**2 + S**2 = I.

R is stored in A(1:K+L,N-K-L+1:N) on exit.
```

If M-K-L < 0,

```
             K M-K K+L-M
D1 =   K ( I  0    0   )
     M-K ( 0  C    0   )

               K M-K K+L-M
D2 =   M-K ( 0  S    0   )
     K+L-M ( 0  0    I   )
```

(continues on next page)

(continued from previous page)

```
       P-L ( 0  0    0   )

           N-K-L  K   M-K  K+L-M
```

(0 R) = K (0 R11 R12 R13)

```
  M-K ( 0     0   R22  R23  )
K+L-M ( 0     0    0   R33  )
```

where C = diag(ALPHA(K+1), ... , ALPHA(M)), S = diag(BETA(K+1), ... , BETA(M)), C**2 + S**2 = I.

R = (R11 R12 R13) is stored in A(1:M, N-K-L+1:N) and R33 is stored

```
(  0  R22 R23 )
```

in B(M-K+1:L,N+M-K-L+1:N) on exit.

The computation of the orthogonal transformation matrices U, V or Q is optional. These matrices may either be formed explicitly, or they may be postmultiplied into input matrices U1, V1, or Q1.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtgsja(JOBU, JOBV, JOBQ, M, P, N, K, L, A, LDA, B, LDB, TOLA, TOLB,
                  ALPHA, BETA, U, LDU, V, LDV, Q, LDQ, WORK, NCYCLE, INFO)
```

C specification:

```
#include "armpl.h"

void dtgsja_(const char *jobu, const char *jobv, const char *jobq,
             const armpl_int_t *m, const armpl_int_t *p, const armpl_int_t *n,
             const armpl_int_t *k, const armpl_int_t *l, double *a,
             const armpl_int_t *lda, double *b, const armpl_int_t *ldb,
             const double *tola, const double *tolb, double *alpha,
             double *beta, double *u, const armpl_int_t *ldu, double *v,
             const armpl_int_t *ldv, double *q, const armpl_int_t *ldq,
             double *work, armpl_int_t *ncycle, armpl_int_t *info, ... );
```

Parameters

JOBU Input parameter.

JOBU is CHARACTER*1

= 'U': U must contain an orthogonal matrix U1 on entry, and the product U1* U is returned; = 'I': U is initialized to the unit matrix, and the orthogonal matrix U is returned; = 'N': U is not computed.

JOBV Input parameter.

JOBV is CHARACTER*1

= 'V': V must contain an orthogonal matrix V1 on entry, and the product V1* V is returned; = 'I': V is initialized to the unit matrix, and the orthogonal matrix V is returned; = 'N': V is not computed.

JOBQ Input parameter.

JOBQ is CHARACTER*1

= 'Q': Q must contain an orthogonal matrix Q1 on entry, and the product Q1* Q is returned; = 'I': Q is initialized to the unit matrix, and the orthogonal matrix Q is returned; = 'N': Q is not computed.

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

P Input parameter.

P is INTEGER

The number of rows of the matrix B. P >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrices A and B. N >= 0.

K Input parameter.

K is INTEGER

L Input parameter.

L is INTEGER

K and L specify the subblocks in the input matrices A and B: A23 = A(K+1:MIN(K+L, M),N-L+1:N) and B13 = B(1:L,N-L+1:N) of A and B, whose GSVD is going to be computed by DTGSJA. See Further Details.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, A(N-K+1:N,1:MIN(K+L, M)) contains the triangular matrix R or part of R. See Purpose for details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). On entry, the P-by-N matrix B. On exit, if necessary, B(M-K+1:L,N+M-K-L+1:N) contains a part of R. See Purpose for details.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, P).

TOLA Input parameter.

TOLA is DOUBLE PRECISION

TOLB Input parameter.

TOLB is DOUBLE PRECISION

TOLA and TOLB are the convergence criteria for the Jacobi- Kogbetliantz iteration procedure. Generally, they are the same as used in the preprocessing step, say TOLA = max(M, N)*norm(A)*MAZHEPS, TOLB = max(P, N)*norm(B)*MAZHEPS.

ALPHA Output parameter.

ALPHA is DOUBLE PRECISION

ALPHA is an array, dimension (N) .

BETA Output parameter.

BETA is DOUBLE PRECISION

BETA is an array, dimension (N). On exit, ALPHA and BETA contain the generalized singular value pairs of A and B; ALPHA(1:K) = 1, BETA(1:K) = 0, and if M-K-L >= 0, ALPHA(K+1:K+L) = diag(C), BETA(K+1:K+L) = diag(S), or if M-K-L < 0, ALPHA(K+1:M)= C, ALPHA(M+1:K+L)= 0 BETA(K+1:M) = S, BETA(M+1:K+L) = 1. Furthermore, if K+L < N, ALPHA(K+L+1:N) = 0 and BETA(K+L+1:N) = 0.

U Input and output parameter.

U is DOUBLE PRECISION

U is an array, dimension (LDU, M). On entry, if JOBU = 'U', U must contain a matrix U1 (usually the orthogonal matrix returned by DGGSVP). On exit, if JOBU = 'I', U contains the orthogonal matrix U; if JOBU = 'U', U contains the product U1*U. If JOBU = 'N', U is not referenced.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U. LDU >= max(1, M) if JOBU = 'U'; LDU >= 1 otherwise.

V Input and output parameter.

V is DOUBLE PRECISION

V is an array, dimension (LDV, P). On entry, if JOBV = 'V', V must contain a matrix V1 (usually the orthogonal matrix returned by DGGSVP). On exit, if JOBV = 'I', V contains the orthogonal matrix V; if JOBV = 'V', V contains the product V1*V. If JOBV = 'N', V is not referenced.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. LDV >= max(1, P) if JOBV = 'V'; LDV >= 1 otherwise.

Q Input and output parameter.

Q is DOUBLE PRECISION

Q is an array, dimension (LDQ, N). On entry, if JOBQ = 'Q', Q must contain a matrix Q1 (usually the orthogonal matrix returned by DGGSVP). On exit, if JOBQ = 'I', Q contains the orthogonal matrix Q; if JOBQ = 'Q', Q contains the product Q1*Q. If JOBQ = 'N', Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, N) if JOBQ = 'Q'; LDQ >= 1 otherwise.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (2*N) .

NCYCLE Output parameter.

NCYCLE is INTEGER

The number of cycles required for convergence.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. = 1: the procedure does not converge after MAXIT cycles.

Related Information

For this routine in other precisions, please see *ctgsja*, *stgsja* and *ztgsja*. It also exists with a native C interface as *LAPACKE_dtgsja*.

4.10.29 sbdsdc

sbdsdc computes the singular value decomposition (SVD) of a real N-by-N (upper or lower) bidiagonal matrix B: B = U * S * VT, using a divide and conquer method, where S is a diagonal matrix with non-negative diagonal elements (the singular values of B), and U and VT are orthogonal matrices of left and right singular vectors, respectively. sbdsdc can be used to compute all singular values, and optionally, singular vectors or singular vectors in compact form.

This code makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none. See SLASD3 for details.

The code currently calls SLASDQ if singular values only are desired. However, it can be slightly modified to compute singular values using the divide and conquer method.

Syntax

Fortran specification:

```
use armpl_library

subroutine sbdsdc(UPLO, COMPQ, N, D, E, U, LDU, VT, LDVT, Q, IQ, WORK, IWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void sbdsdc_(const char *uplo, const char *compq, const armpl_int_t *n,
             float *d, float *e, float *u, const armpl_int_t *ldu, float *vt,
             const armpl_int_t *ldvt, float *q, armpl_int_t *iq, float *work,
             armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': B is upper bidiagonal. = 'L': B is lower bidiagonal.

COMPQ Input parameter.

COMPQ is CHARACTER*1

Specifies whether singular vectors are to be computed as follows: = 'N': Compute singular values only; = 'P': Compute singular values and compute singular vectors in compact form; = 'I': Compute singular values and singular vectors.

N Input parameter.

N is INTEGER

The order of the matrix B. N >= 0.

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry, the n diagonal elements of the bidiagonal matrix B. On exit, if INFO=0, the singular values of B.

E Input and output parameter.

E is REAL

E is an array, dimension (N-1). On entry, the elements of E contain the offdiagonal elements of the bidiagonal matrix whose SVD is desired. On exit, E has been destroyed.

U Output parameter.

U is REAL

U is an array, dimension (LDU, N). If COMPQ = 'I', then: On exit, if INFO = 0, U contains the left singular vectors of the bidiagonal matrix. For other values of COMPQ, U is not referenced.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U. LDU >= 1. If singular vectors are desired, then LDU >= max(1, N).

VT Output parameter.

VT is REAL

VT is an array, dimension (LDVT, N). If COMPQ = 'I', then: On exit, if INFO = 0, VT^T contains the right singular vectors of the bidiagonal matrix. For other values of COMPQ, VT is not referenced.

LDVT Input parameter.

LDVT is INTEGER

The leading dimension of the array VT. LDVT >= 1. If singular vectors are desired, then LDVT >= max(1, N).

Q Output parameter.

Q is REAL

Q is an array, dimension (LDQ). If COMPQ = 'P', then: On exit, if INFO = 0, Q and IQ contain the left and right singular vectors in a compact form, requiring O(N log N) space instead of 2*N**2. In particular, Q contains all the REAL data in LDQ >= N*(11 + 2*SMLSIZ + 8*INT(LOG_2(N/(SMLSIZ+1)))) words of memory, where SMLSIZ is returned by ILAENV and is equal to the maximum size of the subproblems at the bottom of the computation tree (usually about 25). For other values of COMPQ, Q is not referenced.

IQ Output parameter.

IQ is INTEGER array, dimension (LDIQ)

If COMPQ = 'P', then: On exit, if INFO = 0, Q and IQ contain the left and right singular vectors in a compact form, requiring O(N log N) space instead of 2*N**2. In particular, IQ contains all INTEGER data in LDIQ >= N*(3 + 3*INT(LOG_2(N/(SMLSIZ+1)))) words of memory, where SMLSIZ is returned by ILAENV and is equal to the maximum size of the subproblems at the bottom of the computation tree (usually about 25). For other values of COMPQ, IQ is not referenced.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1,LWORK)). If COMPQ = 'N' then LWORK >= (4 * N). If COMPQ = 'P' then LWORK >= (6 * N). If COMPQ = 'I' then LWORK >= (3 * N**2 + 4 * N).

IWORK Output parameter.

IWORK is INTEGER array, dimension (8*N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: The algorithm failed to compute a singular value. The update process of divide and conquer failed.

Related Information

For this routine in other precisions, please see *dbdsdc*. It also exists with a native C interface as *LAPACKE_sbdsdc*.

4.10.30 sbdsqr

sbdsqr computes the singular values and, optionally, the right and/or left singular vectors from the singular value decomposition (SVD) of a real N-by-N (upper or lower) bidiagonal matrix B using the implicit zero-shift QR algorithm. The SVD of B has the form

```
B = Q * S * P**T
```

where S is the diagonal matrix of singular values, Q is an orthogonal matrix of left singular vectors, and P is an orthogonal matrix of right singular vectors. If left singular vectors are requested, this subroutine actually returns U*Q instead of Q, and, if right singular vectors are requested, this subroutine returns P^T *VT instead of P^T , for given real input matrices U and VT. When U and VT are the orthogonal matrices that reduce a general matrix A to bidiagonal form: A = U*B*VT, as computed by SGEBRD, then

```
A = (U*Q) * S * (P**T*VT)
```

is the SVD of A. Optionally, the subroutine may also compute Q^T *C for a given real input matrix C.

See "Computing Small Singular Values of Bidiagonal Matrices With Guaranteed High Relative Accuracy," by J. Demmel and W. Kahan, LAPACK Working Note #3 (or SIAM J. Sci. Statist. Comput. vol. 11, no. 5, pp. 873-912, Sept 1990) and "Accurate singular values and differential qd algorithms," by B. Parlett and V. Fernando, Technical Report CPAM-554, Mathematics Department, University of California at Berkeley, July 1992 for a detailed description of the algorithm.

Syntax

Fortran specification:

```
use armpl_library

subroutine sbdsqr(UPLO, N, NCVT, NRU, NCC, D, E, VT, LDVT, U, LDU, C, LDC,
                  WORK, INFO)
```

C specification:

```
#include "armpl.h"

void sbdsqr_(const char *uplo, const armpl_int_t *n, const armpl_int_t *ncvt,
             const armpl_int_t *nru, const armpl_int_t *ncc, float *d,
             float *e, float *vt, const armpl_int_t *ldvt, float *u,
             const armpl_int_t *ldu, float *c, const armpl_int_t *ldc,
             float *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': B is upper bidiagonal; = 'L': B is lower bidiagonal.

N Input parameter.

N is INTEGER

The order of the matrix B. N >= 0.

NCVT Input parameter.

NCVT is INTEGER

The number of columns of the matrix VT. NCVT >= 0.

NRU Input parameter.

NRU is INTEGER

The number of rows of the matrix U. NRU >= 0.

NCC Input parameter.

NCC is INTEGER

The number of columns of the matrix C. NCC >= 0.

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry, the n diagonal elements of the bidiagonal matrix B. On exit, if INFO=0, the singular values of B in decreasing order.

E Input and output parameter.

E is REAL

E is an array, dimension (N-1). On entry, the N-1 offdiagonal elements of the bidiagonal matrix B. On exit, if INFO = 0, E is destroyed; if INFO > 0, D and E will contain the diagonal and superdiagonal elements of a bidiagonal matrix orthogonally equivalent to the one given as input.

VT Input and output parameter.

VT is REAL

VT is an array, dimension (LDVT, NCVT). On entry, an N-by-NCVT matrix VT. On exit, VT is overwritten by P^T * VT. Not referenced if NCVT = 0.

LDVT Input parameter.

LDVT is INTEGER

The leading dimension of the array VT. LDVT >= max(1, N) if NCVT > 0; LDVT >= 1 if NCVT = 0.

U Input and output parameter.

U is REAL

U is an array, dimension (LDU, N). On entry, an NRU-by-N matrix U. On exit, U is overwritten by U * Q. Not referenced if NRU = 0.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U. LDU >= max(1, NRU).

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, NCC). On entry, an N-by-NCC matrix C. On exit, C is overwritten by Q^T * C. Not referenced if NCC = 0.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, N) if NCC > 0; LDC >=1 if NCC = 0.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (4*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: If INFO = -i, the i-th argument had an illegal value > 0: if NCVT = NRU = NCC = 0, = 1, a split was marked by a positive value in E = 2, current block of Z not diagonalized after 30*N iterations (in inner while loop) = 3, termination criterion of outer while loop not met (program created more than N unreduced blocks) else NCVT = NRU = NCC = 0, the algorithm did not converge; D and E contain the elements of a bidiagonal matrix which is orthogonally similar to the input matrix B; if INFO = i, i elements of E have not converged to zero.

Related Information

For this routine in other precisions, please see *cbdsqr*, *dbdsqr* and *zbdsqr*. It also exists with a native C interface as *LAPACKE_sbdsqr*.

4.10.31 sbdsvdx

```
SBDSVDX computes the singular value decomposition (SVD) of a real
 N-by-N (upper or lower) bidiagonal matrix B, B = U * S * VT,
 where S is a diagonal matrix with non-negative diagonal elements
 (the singular values of B), and U and VT are orthogonal matrices
 of left and right singular vectors, respectively.

 Given an upper bidiagonal B with diagonal D = [ d_1 d_2 ... d_N ]
 and superdiagonal E = [ e_1 e_2 ... e_N-1 ], SBDSVDX computes the
 singular value decompositon of B through the eigenvalues and
 eigenvectors of the N*2-by-N*2 tridiagonal matrix

       |  0  d_1               |
       | d_1  0  e_1           |
 TGK = |     e_1  0  d_2       |
       |         d_2  .   .    |
       |              .   .  . |

 If (s,u,v) is a singular triplet of B with ||u|| = ||v|| = 1, then
 (+/-s,q), ||q|| = 1, are eigenpairs of TGK, with q = P * ( u' +/-v' ) /
 sqrt(2) = ( v_1 u_1 v_2 u_2 ... v_n u_n ) / sqrt(2), and
 P = [ e_{n+1} e_{1} e_{n+2} e_{2} ... ].

 Given a TGK matrix, one can either a) compute -s,-v and change signs
 so that the singular values (and corresponding vectors) are already in
 descending order (as in SGESVD/SGESDD) or b) compute s,v and reorder
 the values (and corresponding vectors). SBDSVDX implements a) by
 calling SSTEVX (bisection plus inverse iteration, to be replaced
 with a version of the Multiple Relative Robust Representation
 algorithm. (See P. Willems and B. Lang, A framework for the MR^3
 algorithm: theory and implementation, SIAM J. Sci. Comput.,
 35:740-766, 2013.)
```

Syntax

Fortran specification:

```
use armpl_library

subroutine sbdsvdx(UPLO, JOBZ, RANGE, N, D, E, VL, VU, IL, IU, NS, S, Z, LDZ,
                   WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sbdsvdx_(const char *uplo, const char *jobz, const char *range,
              const armpl_int_t *n, const float *d, const float *e,
              const float *vl, const float *vu, const armpl_int_t *il,
              const armpl_int_t *iu, armpl_int_t *ns, float *s, float *z,
              const armpl_int_t *ldz, float *work, armpl_int_t *iwork,
              armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': B is upper bidiagonal; = 'L': B is lower bidiagonal.

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute singular values only; = 'V': Compute singular values and singular vectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all singular values will be found. = 'V': all singular values in the half-open interval [VL, VU) will be found. = 'I': the IL-th through IU-th singular values will be found.

N Input parameter.

N is INTEGER

The order of the bidiagonal matrix. N >= 0.

D Input parameter.

D is REAL

D is an array, dimension (N). The n diagonal elements of the bidiagonal matrix B.

E Input parameter.

E is REAL

E is an array, dimension (max(1,N-1)). The (n-1) superdiagonal elements of the bidiagonal matrix B in elements 1 to N-1.

VL Input parameter.

VL is REAL

If RANGE='V', the lower bound of the interval to be searched for singular values. VU > VL. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is REAL

If RANGE='V', the upper bound of the interval to be searched for singular values. VU > VL. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest singular value to be returned. 1 <= IL <= IU <= min(M, N), if min(M, N) > 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest singular value to be returned. 1 <= IL <= IU <= min(M, N), if min(M, N) > 0. Not referenced if RANGE = 'A' or 'V'.

NS Output parameter.

NS is INTEGER

The total number of singular values found. 0 <= NS <= N. If RANGE = 'A', NS = N, and if RANGE = 'I', NS = IU-IL+1.

S Output parameter.

S is REAL

S is an array, dimension (N). The first NS elements contain the selected singular values in ascending order.

Z Output parameter.

Z is REAL

Z is an array, dimension (2*N,K)). If JOBZ = 'V', then if INFO = 0 the first NS columns of Z contain the singular vectors of the matrix B corresponding to the selected singular values, with U in rows 1 to N and V in rows N+1 to N*2, i.e. Z = [U] [V] If JOBZ = 'N', then Z is not referenced. Note: The user must ensure that at least K = NS+1 columns are supplied in the array Z; if RANGE = 'V', the exact value of NS is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(2,N*2).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (14*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (12*N)

If JOBZ = 'V', then if INFO = 0, the first NS elements of IWORK are zero. If INFO > 0, then IWORK contains the indices of the eigenvectors that failed to converge in DSTEVX.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, then i eigenvectors failed to converge in SSTEVX. The indices of the eigenvectors (as returned by SSTEVX) are stored in the array IWORK. if INFO = N*2 + 1, an internal error occurred.

Related Information

For this routine in other precisions, please see *dbdsvdx*. It also exists with a native C interface as *LAPACKE_sbdsvdx*.

4.10.32 sgbbrd

`sgbbrd` reduces a real general m-by-n band matrix A to upper bidiagonal form B by an orthogonal transformation: Q^T * A * P = B.

The routine computes B, and optionally forms Q or P^T , or computes Q^T *C for a given matrix C.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgbbrd(VECT, M, N, NCC, KL, KU, AB, LDAB, D, E, Q, LDQ, PT, LDPT,
                  C, LDC, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgbbrd_(const char *vect, const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *ncc, const armpl_int_t *kl,
             const armpl_int_t *ku, float *ab, const armpl_int_t *ldab,
             float *d, float *e, float *q, const armpl_int_t *ldq, float *pt,
             const armpl_int_t *ldpt, float *c, const armpl_int_t *ldc,
             float *work, armpl_int_t *info, ... );
```

Parameters

VECT Input parameter.

`VECT` is `CHARACTER*1`

Specifies whether or not the matrices `Q` and P^T are to be formed. = 'N': do not form `Q` or P^T ; = 'Q': form `Q` only; = 'P': form P^T only; = 'B': form both.

M Input parameter.

`M` is `INTEGER`

The number of rows of the matrix A. `M` >= 0.

N Input parameter.

`N` is `INTEGER`

The number of columns of the matrix A. `N` >= 0.

NCC Input parameter.

`NCC` is `INTEGER`

The number of columns of the matrix `C`. `NCC` >= 0.

KL Input parameter.

`KL` is `INTEGER`

The number of subdiagonals of the matrix A. `KL` >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals of the matrix A. KU >= 0.

AB Input and output parameter.

AB is REAL

AB is an array, dimension (LDAB, N). On entry, the m-by-n band matrix A, stored in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(ku+1+i-j,j) = A(i,j) for max(1,j-ku)<=i<=min(m,j+kl). On exit, A is overwritten by values generated during the reduction.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array A. LDAB >= KL+KU+1.

D Output parameter.

D is REAL

D is an array, dimension (min(M, N)). The diagonal elements of the bidiagonal matrix B.

E Output parameter.

E is REAL

E is an array, dimension (min(M, N)-1). The superdiagonal elements of the bidiagonal matrix B.

Q Output parameter.

Q is REAL

Q is an array, dimension (LDQ, M). If VECT = 'Q' or 'B', the m-by-m orthogonal matrix Q. If VECT = 'N' or 'P', the array Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, M) if VECT = 'Q' or 'B'; LDQ >= 1 otherwise.

PT Output parameter.

PT is REAL

PT is an array, dimension (LDPT, N). If VECT = 'P' or 'B', the n-by-n orthogonal matrix P'. If VECT = 'N' or 'Q', the array PT is not referenced.

LDPT Input parameter.

LDPT is INTEGER

The leading dimension of the array PT. LDPT >= max(1, N) if VECT = 'P' or 'B'; LDPT >= 1 otherwise.

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, NCC). On entry, an m-by-ncc matrix C. On exit, C is overwritten by Q^T *C. C is not referenced if NCC = 0.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M) if NCC > 0; LDC >= 1 if NCC = 0.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (2*max(M, N)) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cgbbrd*, *dgbbrd* and *zgbbrd*. It also exists with a native C interface as *LAPACKE_sgbbrd*.

4.10.33 sgebrd

sgebrd reduces a general real M-by-N matrix A to upper or lower bidiagonal form B by an orthogonal transformation: $Q^T * A * P = B$.

If m >= n, B is upper bidiagonal; if m < n, B is lower bidiagonal.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgebrd(M, N, A, LDA, D, E, TAUQ, TAUP, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgebrd_(const armpl_int_t *m, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, float *d, float *e, float *tauq,
             float *taup, float *work, const armpl_int_t *lwork,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows in the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns in the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the M-by-N general matrix to be reduced. On exit, if m >= n, the diagonal and the first superdiagonal are overwritten with the upper bidiagonal matrix B; the elements below the diagonal, with the array TAUQ, represent the orthogonal matrix Q as a product of elementary reflectors, and the elements above the first superdiagonal, with the array TAUP, represent the orthogonal matrix P as a product of elementary reflectors; if m < n, the diagonal and the first subdiagonal are overwritten with the lower bidiagonal matrix B; the elements below the first subdiagonal, with the array TAUQ, represent the orthogonal matrix Q as a product of elementary reflectors, and the elements above the diagonal, with the array TAUP, represent the orthogonal matrix P as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

D Output parameter.

D is REAL

D is an array, dimension (min(M, N)). The diagonal elements of the bidiagonal matrix B: D(i) = A(i,i).

E Output parameter.

E is REAL

E is an array, dimension (min(M, N)-1). The off-diagonal elements of the bidiagonal matrix B: if m >= n, E(i) = A(i,i+1) for i = 1,2,...,n-1; if m < n, E(i) = A(i+1,i) for i = 1,2,...,m-1.

TAUQ Output parameter.

TAUQ is REAL

TAUQ is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors which represent the orthogonal matrix Q. See Further Details.

TAUP Output parameter.

TAUP is REAL

TAUP is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors which represent the orthogonal matrix P. See Further Details.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= max(1, M, N). For optimum performance LWORK >= (M+N)*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cgebrd*, *dgebrd* and *zgebrd*. It also exists with a native C interface as *LAPACKE_sgebrd*.

4.10.34 sgejsv

sgejsv computes the singular value decomposition (SVD) of a real M-by-N matrix [A], where M >= N. The SVD of [A] is written as

```
[A] = [U] * [SIGMA] * [V]^t,
```

where [SIGMA] is an N-by-N (M-by-N) matrix which is zero except for its N diagonal elements, [U] is an M-by-N (or M-by-M) orthonormal matrix, and [V] is an N-by-N orthogonal matrix. The diagonal elements of [SIGMA] are the singular values of [A]. The columns of [U] and [V] are the left and the right singular vectors of [A], respectively. The matrices [U] and [V] are computed and stored in the arrays U and V, respectively. The diagonal of [SIGMA] is computed and stored in the array SVA. `sgejsv` can sometimes compute tiny singular values and their singular vectors much more accurately than other SVD routines, see below under Further Details.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgejsv(JOBA, JOBU, JOBV, JOBR, JOBT, JOBP, M, N, A, LDA, SVA, U,
                  LDU, V, LDV, WORK, LWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgejsv_(const char *joba, const char *jobu, const char *jobv,
             const char *jobr, const char *jobt, const char *jobp,
             const armpl_int_t *m, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, float *sva, float *u,
             const armpl_int_t *ldu, float *v, const armpl_int_t *ldv,
             float *work, const armpl_int_t *lwork, armpl_int_t *iwork,
             armpl_int_t *info, ... );
```

Parameters

JOBA Input parameter.

JOBA is CHARACTER*1

Specifies the level of accuracy: = 'C': This option works well (high relative accuracy) if A = B * D, with well-conditioned B and arbitrary diagonal matrix D. The accuracy cannot be spoiled by COLUMN scaling. The accuracy of the computed output depends on the condition of B, and the procedure aims at the best theoretical accuracy. The relative error max_{i=1:N}|d sigma_i| / sigma_i is bounded by f(M, N)*epsilon* cond(B), independent of D. The input matrix is preprocessed with the QRF with column pivoting. This initial preprocessing and preconditioning by a rank revealing QR factorization is common for all values of JOBA. Additional actions are specified as follows: = 'E': Computation as with 'C' with an additional estimate of the condition number of B. It provides a realistic error bound. = 'F': If A = D1 * C * D2 with ill-conditioned diagonal scalings D1, D2, and well-conditioned matrix C, this option gives higher accuracy than the 'C' option. If the structure of the input matrix is not known, and relative accuracy is desirable, then this option is advisable. The input matrix A is preprocessed with QR factorization with FULL (row and column) pivoting. = 'G' Computation as with 'F' with an additional estimate of the condition number of B, where A=D*B. If A has heavily weighted rows, then using this condition number gives too pessimistic error bound. = 'A': Small singular values are the noise and the matrix is treated as numerically rank deficient. The error in the computed singular values is bounded by f(m,n)*epsilon*||A||. The computed SVD A = U * S * V^t restores A up to f(m,n)*epsilon*||A||. This gives the procedure the licence to discard (set to zero) all singular values below N*epsilon*||A||. = 'R': Similar as in 'A'. Rank revealing property of the initial QR factorization is used do reveal (using triangular factor) a gap sigma_{r+1} < epsilon * sigma_r in which case the numerical RANK is declared to be r. The SVD is computed with absolute error bounds, but more accurately than with 'A'.

JOBU Input parameter.

JOBU is CHARACTER*1

Specifies whether to compute the columns of U: = 'U': N columns of U are returned in the array U. = 'F': full set of M left sing. vectors is returned in the array U. = 'W': U may be used as workspace of length M*N. See the description of U. = 'N': U is not computed.

JOBV Input parameter.

JOBV is CHARACTER*1

Specifies whether to compute the matrix V: = 'V': N columns of V are returned in the array V; Jacobi rotations are not explicitly accumulated. = 'J': N columns of V are returned in the array V, but they are computed as the product of Jacobi rotations. This option is allowed only if JOBU .NE. 'N', i.e. in computing the full SVD. = 'W': V may be used as workspace of length N*N. See the description of V. = 'N': V is not computed.

JOBR Input parameter.

JOBR is CHARACTER*1

Specifies the RANGE for the singular values. Issues the licence to set to zero small positive singular values if they are outside specified range. If A .NE. 0 is scaled so that the largest singular value of c* A is around SQRT(BIG), BIG=SLAMCH('O'), then JOBR issues the licence to kill columns of A whose norm in c* A is less than SQRT(SFMIN) (for JOBR.EQ.'R'), or less than SMALL=SFMIN/EPSLN, where SFMIN=SLAMCH('S'), EPSLN=SLAMCH('E'). = 'N': Do not kill small columns of c*A. This option assumes that BLAS and QR factorizations and triangular solvers are implemented to work in that range. If the condition of A is greater than BIG, use SGESVJ. = 'R': RESTRICTED range for sigma(c*A) is [SQRT(SFMIN), SQRT(BIG)] (roughly, as described above). This option is recommended. ========================== For computing the singular values in the FULL range [SFMIN,BIG] use SGESVJ.

JOBT Input parameter.

JOBT is CHARACTER*1

If the matrix is square then the procedure may determine to use transposed A if A^t seems to be better with respect to convergence. If the matrix is not square, JOBT is ignored. This is subject to changes in the future. The decision is based on two values of entropy over the adjoint orbit of A^t * A. See the descriptions of WORK(6) and WORK(7). = 'T': transpose if entropy test indicates possibly faster convergence of Jacobi process if A^t is taken as input. If A is replaced with A^t, then the row pivoting is included automatically. = 'N': do not speculate. This option can be used to compute only the singular values, or the full SVD (U, SIGMA and V). For only one set of singular vectors (U or V), the caller should provide both U and V, as one of the matrices is used as workspace if the matrix A is transposed. The implementer can easily remove this constraint and make the code more complicated. See the descriptions of U and V.

JOBP Input parameter.

JOBP is CHARACTER*1

Issues the licence to introduce structured perturbations to drown denormalized numbers. This licence should be active if the denormals are poorly implemented, causing slow computation, especially in cases of fast convergence (!). For details see [1,2]. For the sake of simplicity, this perturbations are included only when the full SVD or only the singular values are requested. The implementer/user can easily add the perturbation for the cases of computing one set of singular vectors. = 'P': introduce perturbation = 'N': do not perturb

M Input parameter.

M is INTEGER

The number of rows of the input matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the input matrix A. M >= N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

SVA Output parameter.

SVA is REAL

SVA is an array, dimension (N). On exit, - For WORK(1)/WORK(2) = ONE: The singular values of A. During the computation SVA contains Euclidean column norms of the iterated matrices in the array A. - For WORK(1) .NE. WORK(2): The singular values of A are (WORK(1)/WORK(2)) * SVA(1:N). This factored form is used if sigma_max(A) overflows or if small singular values have been saved from underflow by scaling the input matrix A. - If JOBR='R' then some of the singular values may be returned as exact zeros obtained by "set to zero" because they are below the numerical rank threshold or are denormalized numbers.

U Output parameter.

U is REAL

U is an array, dimension (LDU, N). If JOBU = 'U', then U contains on exit the M-by-N matrix of the left singular vectors. If JOBU = 'F', then U contains on exit the M-by-M matrix of the left singular vectors, including an ONB of the orthogonal complement of the Range(A). If JOBU = 'W' .AND. (JOBV.EQ.'V' .AND. JOBT.EQ.'T' .AND. M.EQ.N), then U is used as workspace if the procedure replaces A with A^t. In that case, [V] is computed in U as left singular vectors of A^t and then copied back to the V array. This 'W' option is just a reminder to the caller that in this case U is reserved as workspace of length N*N. If JOBU = 'N' U is not referenced, unless JOBT='T'.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U, LDU >= 1. IF JOBU = 'U' or 'F' or 'W', then LDU >= M.

V Output parameter.

V is REAL

V is an array, dimension (LDV, N). If JOBV = 'V', 'J' then V contains on exit the N-by-N matrix of the right singular vectors; If JOBV = 'W', AND (JOBU.EQ.'U' AND JOBT.EQ.'T' AND M.EQ.N), then V is used as workspace if the pprocedure replaces A with A^t. In that case, [U] is computed in V as right singular vectors of A^t and then copied back to the U array. This 'W' option is just a reminder to the caller that in this case V is reserved as workspace of length N*N. If JOBV = 'N' V is not referenced, unless JOBT='T'.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V, LDV >= 1. If JOBV = 'V' or 'J' or 'W', then LDV >= N.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (LWORK). On exit, WORK(1) = SCALE = WORK(2) / WORK(1) is the scaling factor such that SCALE*SVA(1:N) are the computed singular values of A. (See the description of SVA().) WORK(2) = See the description of WORK(1). WORK(3) = SCONDA is an estimate for the condition number of column equilibrated A. (If JOBA .EQ. 'E' or 'G') SCONDA is an estimate of SQRT(||(R^t * R)^(-1)||_1). It is computed using SPOCON. It holds N^(-1/4) * SCONDA <= ||R^(-1)||_2 <= N^(1/4) * SCONDA where R is the triangular factor from the QRF of A. However, if R is truncated and the numerical rank is determined to be strictly smaller than N, SCONDA is returned as -1, thus indicating that the smallest singular values might be lost.

If full SVD is needed, the following two condition numbers are useful for the analysis of the algorithm. They are provied for a developer/implementer who is familiar with the details of the method.

WORK(4) = an estimate of the scaled condition number of the triangular factor in the first QR factorization. WORK(5) = an estimate of the scaled condition number of the triangular factor in the second QR factorization.

The following two parameters are computed if JOBT .EQ. 'T'. They are provided for a developer/implementer who is familiar with the details of the method.

WORK(6) = the entropy of A^t* A :: this is the Shannon entropy of diag(A^t*A) / Trace(A^t*A) taken as point in the probability simplex. WORK(7) = the entropy of A*A^t.

LWORK Input parameter.

LWORK is INTEGER

Length of WORK to confirm proper allocation of work space. LWORK depends on the job:

If only SIGMA is needed (JOBU.EQ.'N', JOBV.EQ.'N') and -> .. no scaled condition estimate required (JOBE.EQ.'N'): LWORK >= max(2*M+N,4*N+1,7). This is the minimal requirement. ->> For optimal performance (blocked code) the optimal value is LWORK >= max(2*M+N,3*N+(N+1)*NB,7). Here NB is the optimal block size for DGEQP3 and DGEQRF. In general, optimal LWORK is computed as LWORK >= max(2*M+N,N+LWORK(DGEQP3),N+LWORK(DGEQRF), 7). -> .. an estimate of the scaled condition number of A is required (JOBA='E', 'G'). In this case, LWORK is the maximum of the above and N*N+4*N, i.e. LWORK >= max(2*M+N,N*N+4*N,7). ->> For optimal performance (blocked code) the optimal value is LWORK >= max(2*M+N,3*N+(N+1)*NB, N*N+4*N, 7). In general, the optimal length LWORK is computed as LWORK >= max(2*M+N,N+LWORK(DGEQP3),N+LWORK(DGEQRF), N+N*N+LWORK(DPOCON),7).

If SIGMA and the right singular vectors are needed (JOBV.EQ.'V'), -> the minimal requirement is LWORK >= max(2*M+N,4*N+1,7). -> For optimal performance, LWORK >= max(2*M+N,3*N+(N+1)*NB,7), where NB is the optimal block size for DGEQP3, DGEQRF, DGELQ, DORMLQ. In general, the optimal length LWORK is computed as LWORK >= max(2*M+N,N+LWORK(DGEQP3), N+LWORK(DPOCON), N+LWORK(DGELQ), 2*N+LWORK(DGEQRF), N+LWORK(DORMLQ)).

If SIGMA and the left singular vectors are needed -> the minimal requirement is LWORK >= max(2*M+N,4*N+1,7). -> For optimal performance: if JOBU.EQ.'U' :: LWORK >= max(2*M+N,3*N+(N+1)*NB,7), if JOBU.EQ.'F' :: LWORK >= max(2*M+N,3*N+(N+1)*NB,N+M*NB,7), where NB is the optimal block size for DGEQP3, DGEQRF, DORMQR. In general, the optimal length LWORK is computed as LWORK >= max(2*M+N,N+LWORK(DGEQP3),N+LWORK(DPOCON), 2*N+LWORK(DGEQRF), N+LWORK(DORMQR)). Here LWORK(DORMQR) equals N*NB (for JOBU.EQ.'U') or M*NB (for JOBU.EQ.'F').

If the full SVD is needed: (JOBU.EQ.'U' or JOBU.EQ.'F') and -> if JOBV.EQ.'V' the minimal requirement is LWORK >= max(2*M+N,6*N+2*N*N). -> if JOBV.EQ.'J' the minimal requirement is LWORK >= max(2*M+N, 4*N+N*N,2*N+N*N+6). -> For optimal performance, LWORK should be additionally larger than N+M*NB, where NB is the optimal block size for DORMQR.

IWORK Output parameter.

IWORK is INTEGER array, dimension (M+3*N).

On exit, IWORK(1) = the numerical rank determined after the initial QR factorization with pivoting. See the descriptions of JOBA and JOBR. IWORK(2) = the number of the computed nonzero singular values IWORK(3) = if nonzero, a warning message: If IWORK(3).EQ.1 then some of the column norms of A were denormalized floats. The requested high accuracy is not warranted by the data.

INFO Output parameter.

INFO is INTEGER

< 0 : if INFO = -i, then the i-th argument had an illegal value. = 0 : successful exit; > 0 : SGEJSV did not converge in the maximal allowed number of sweeps. The computed values may be inaccurate.

Related Information

For this routine in other precisions, please see *cgejsv*, *dgejsv* and *zgejsv*. It also exists with a native C interface as *LAPACKE_sgejsv*.

4.10.35 sgesdd

`sgesdd` computes the singular value decomposition (SVD) of a real M-by-N matrix A, optionally computing the left and right singular vectors. If singular vectors are desired, it uses a divide-and-conquer algorithm.

The SVD is written

```
A = U * SIGMA * transpose(V)
```

where SIGMA is an M-by-N matrix which is zero except for its min(m,n) diagonal elements, U is an M-by-M orthogonal matrix, and V is an N-by-N orthogonal matrix. The diagonal elements of SIGMA are the singular values of A; they are real and non-negative, and are returned in descending order. The first min(m,n) columns of U and V are the left and right singular vectors of A.

Note that the routine returns VT = V^T , not V.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgesdd(JOBZ, M, N, A, LDA, S, U, LDU, VT, LDVT, WORK, LWORK, IWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void sgesdd_(const char *jobz, const armpl_int_t *m, const armpl_int_t *n,
             float *a, const armpl_int_t *lda, float *s, float *u,
             const armpl_int_t *ldu, float *vt, const armpl_int_t *ldvt,
             float *work, const armpl_int_t *lwork, armpl_int_t *iwork,
             armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

Specifies options for computing all or part of the matrix U: = 'A': all M columns of U and all N rows of V^T are returned in the arrays U and VT; = 'S': the first min(M, N) columns of U and the first min(M, N) rows of V^T are returned in the arrays U and VT; = 'O': If M >= N, the first N columns of U are overwritten on the array A and all rows of V^T are returned in the array VT; otherwise, all columns of U are returned in the array U and the first M rows of V^T are overwritten in the array A; = 'N': no columns of U or rows of V^T are computed.

M Input parameter.

M is INTEGER

The number of rows of the input matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the input matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, if JOBZ = 'O', A is overwritten with the first N columns of U (the left singular vectors, stored columnwise) if M >= N; A is overwritten with the first M rows of V^T (the right singular vectors, stored rowwise) otherwise. if JOBZ .ne. 'O', the contents of A are destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

S Output parameter.

S is REAL

S is an array, dimension (min(M, N)). The singular values of A, sorted so that S(i) >= S(i+1).

U Output parameter.

U is REAL

U is an array, dimension (LDU,UCOL). UCOL = M if JOBZ = 'A' or JOBZ = 'O' and M < N; UCOL = min(M, N) if JOBZ = 'S'. If JOBZ = 'A' or JOBZ = 'O' and M < N, U contains the M-by-M orthogonal matrix U; if JOBZ = 'S', U contains the first min(M, N) columns of U (the left singular vectors, stored columnwise); if JOBZ = 'O' and M >= N, or JOBZ = 'N', U is not referenced.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U. LDU >= 1; if JOBZ = 'S' or 'A' or JOBZ = 'O' and M < N, LDU >= M.

VT Output parameter.

VT is REAL

VT is an array, dimension (LDVT, N). If JOBZ = 'A' or JOBZ = 'O' and M >= N, VT contains the N-by-N orthogonal matrix V^T ; if JOBZ = 'S', VT contains the first min(M, N) rows of V^T (the right singular vectors, stored rowwise); if JOBZ = 'O' and M < N, or JOBZ = 'N', VT is not referenced.

LDVT Input parameter.

LDVT is INTEGER

The leading dimension of the array VT. LDVT >= 1; if JOBZ = 'A' or JOBZ = 'O' and M >= N, LDVT >= N; if JOBZ = 'S', LDVT >= min(M, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK;

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= 1. If LWORK = -1, a workspace query is assumed. The optimal size for the WORK array is calculated and stored in WORK(1), and no other work except argument checking is performed.

Let mx = max(M, N) and mn = min(M, N). If JOBZ = 'N', LWORK >= 3*mn + max(mx, 7*mn). If JOBZ = 'O', LWORK >= 3*mn + max(mx, 5*mn*mn + 4*mn). If JOBZ = 'S', LWORK >= 4*mn*mn + 7*mn. If JOBZ = 'A', LWORK >= 4*mn*mn + 6*mn + mx. These are not tight minimums in all cases; see comments inside code. For good performance, LWORK should generally be larger; a query is recommended.

IWORK Output parameter.

IWORK is INTEGER array, dimension (8*min(M, N))

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: SBDSDC did not converge, updating process failed.

Related Information

For this routine in other precisions, please see *cgesdd*, *dgesdd* and *zgesdd*. It also exists with a native C interface as *LAPACKE_sgesdd*.

4.10.36 sgesvd

sgesvd computes the singular value decomposition (SVD) of a real M-by-N matrix A, optionally computing the left and/or right singular vectors. The SVD is written

```
A = U * SIGMA * transpose(V)
```

where SIGMA is an M-by-N matrix which is zero except for its min(m,n) diagonal elements, U is an M-by-M orthogonal matrix, and V is an N-by-N orthogonal matrix. The diagonal elements of SIGMA are the singular values of A; they are real and non-negative, and are returned in descending order. The first min(m,n) columns of U and V are the left and right singular vectors of A.

Note that the routine returns V^T , not V.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgesvd(JOBU, JOBVT, M, N, A, LDA, S, U, LDU, VT, LDVT, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void sgesvd_(const char *jobu, const char *jobvt, const armpl_int_t *m,
             const armpl_int_t *n, float *a, const armpl_int_t *lda, float *s,
             float *u, const armpl_int_t *ldu, float *vt,
             const armpl_int_t *ldvt, float *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

JOBU Input parameter.

JOBU is CHARACTER*1

Specifies options for computing all or part of the matrix U: = 'A': all M columns of U are returned in array U: = 'S': the first min(m,n) columns of U (the left singular vectors) are returned in the array U; = 'O': the first min(m,n) columns of U (the left singular vectors) are overwritten on the array A; = 'N': no columns of U (no left singular vectors) are computed.

JOBVT Input parameter.

JOBVT is CHARACTER*1

Specifies options for computing all or part of the matrix V^T : = 'A': all N rows of V^T are returned in the array VT; = 'S': the first min(m,n) rows of V^T (the right singular vectors) are returned in the array VT; = 'O': the first min(m,n) rows of V^T (the right singular vectors) are overwritten on the array A; = 'N': no rows of V^T (no right singular vectors) are computed.

JOBVT and JOBU cannot both be 'O'.

M Input parameter.

M is INTEGER

The number of rows of the input matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the input matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, if JOBU = 'O', A is overwritten with the first min(m,n) columns of U (the left singular vectors, stored columnwise); if JOBVT = 'O', A is overwritten with the first min(m,n) rows of V^T (the right singular vectors, stored rowwise); if JOBU .ne. 'O' and JOBVT .ne. 'O', the contents of A are destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

S Output parameter.

S is REAL

S is an array, dimension (min(M, N)). The singular values of A, sorted so that S(i) >= S(i+1).

U Output parameter.

U is REAL

U is an array, dimension (LDU,UCOL). (LDU, M) if JOBU = 'A' or (LDU,min(M, N)) if JOBU = 'S'. If JOBU = 'A', U contains the M-by-M orthogonal matrix U; if JOBU = 'S', U contains the first min(m,n) columns of U (the left singular vectors, stored columnwise); if JOBU = 'N' or 'O', U is not referenced.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U. LDU >= 1; if JOBU = 'S' or 'A', LDU >= M.

VT Output parameter.

VT is REAL

VT is an array, dimension (LDVT, N). If JOBVT = 'A', VT contains the N-by-N orthogonal matrix V^T ; if JOBVT = 'S', VT contains the first min(m,n) rows of V^T (the right singular vectors, stored rowwise); if JOBVT = 'N' or 'O', VT is not referenced.

LDVT Input parameter.

LDVT is INTEGER

The leading dimension of the array VT. LDVT >= 1; if JOBVT = 'A', LDVT >= N; if JOBVT = 'S', LDVT >= min(M, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK; if INFO > 0, WORK(2:MIN(M, N)) contains the unconverged superdiagonal elements of an upper bidiagonal matrix B whose diagonal is in S (not necessarily sorted). B satisfies A = U * B * VT, so it has the same singular values as A, and singular vectors related by U and VT.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= MAX(1,5*MIN(M, N)) for the paths (see comments inside code): - PATH 1 (M much larger than N, JOBU='N') - PATH 1t (N much larger than M, JOBVT='N') LWORK >= MAX(1,3*MIN(M, N)+MAX(M, N),5*MIN(M, N)) for the other paths For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if SBDSQR did not converge, INFO specifies how many superdiagonals of an intermediate bidiagonal form B did not converge to zero. See the description of WORK above for details.

Related Information

For this routine in other precisions, please see *cgesvd*, *dgesvd* and *zgesvd*. It also exists with a native C interface as *LAPACKE_sgesvd*.

4.10.37 sgesvdx

```
SGESVDX computes the singular value decomposition (SVD) of a real
 M-by-N matrix A, optionally computing the left and/or right singular
 vectors. The SVD is written

     A = U * SIGMA * transpose(V)

 where SIGMA is an M-by-N matrix which is zero except for its
 min(m,n) diagonal elements, U is an M-by-M orthogonal matrix, and
 V is an N-by-N orthogonal matrix.  The diagonal elements of SIGMA
 are the singular values of A; they are real and non-negative, and
 are returned in descending order.  The first min(m,n) columns of
 U and V are the left and right singular vectors of A.

 SGESVDX uses an eigenvalue problem for obtaining the SVD, which
 allows for the computation of a subset of singular values and
 vectors. See SBDSVDX for details.

 Note that the routine returns V**T, not V.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine sgesvdx(JOBU, JOBVT, RANGE, M, N, A, LDA, VL, VU, IL, IU, NS, S, U,
                   LDU, VT, LDVT, WORK, LWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgesvdx_(const char *jobu, const char *jobvt, const char *range,
              const armpl_int_t *m, const armpl_int_t *n, float *a,
              const armpl_int_t *lda, const float *vl, const float *vu,
              const armpl_int_t *il, const armpl_int_t *iu, armpl_int_t *ns,
              float *s, float *u, const armpl_int_t *ldu, float *vt,
              const armpl_int_t *ldvt, float *work, const armpl_int_t *lwork,
              armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

JOBU Input parameter.

JOBU is CHARACTER*1

Specifies options for computing all or part of the matrix U: = 'V': the first min(m,n) columns of U (the left singular vectors) or as specified by RANGE are returned in the array U; = 'N': no columns of U (no left singular vectors) are computed.

JOBVT Input parameter.

JOBVT is CHARACTER*1

Specifies options for computing all or part of the matrix V^T : = 'V': the first min(m,n) rows of V^T (the right singular vectors) or as specified by RANGE are returned in the array VT; = 'N': no rows of V^T (no right singular vectors) are computed.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all singular values will be found. = 'V': all singular values in the half-open interval (VL,VU] will be found. = 'I': the IL-th through IU-th singular values will be found.

M Input parameter.

M is INTEGER

The number of rows of the input matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the input matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the contents of A are destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

VL Input parameter.

VL is REAL

If RANGE='V', the lower bound of the interval to be searched for singular values. VU > VL. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is REAL

If RANGE='V', the upper bound of the interval to be searched for singular values. VU > VL. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest singular value to be returned. 1 <= IL <= IU <= min(M, N), if min(M, N) > 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest singular value to be returned. 1 <= IL <= IU <= min(M, N), if min(M, N) > 0. Not referenced if RANGE = 'A' or 'V'.

NS Output parameter.

NS is INTEGER

The total number of singular values found, 0 <= NS <= min(M, N). If RANGE = 'A', NS = min(M, N); if RANGE = 'I', NS = IU-IL+1.

S Output parameter.

S is REAL

S is an array, dimension (min(M, N)). The singular values of A, sorted so that S(i) >= S(i+1).

U Output parameter.

U is REAL

U is an array, dimension (LDU,UCOL). If JOBU = 'V', U contains columns of U (the left singular vectors, stored columnwise) as specified by RANGE; if JOBU = 'N', U is not referenced. Note: The user must ensure that UCOL >= NS; if RANGE = 'V', the exact value of NS is not known in advance and an upper bound must be used.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U. LDU >= 1; if JOBU = 'V', LDU >= M.

VT Output parameter.

VT is REAL

VT is an array, dimension (LDVT, N). If JOBVT = 'V', VT contains the rows of V^T (the right singular vectors, stored rowwise) as specified by RANGE; if JOBVT = 'N', VT is not referenced. Note: The user must ensure that LDVT >= NS; if RANGE = 'V', the exact value of NS is not known in advance and an upper bound must be used.

LDVT Input parameter.

LDVT is INTEGER

The leading dimension of the array VT. LDVT >= 1; if JOBVT = 'V', LDVT >= NS (see above).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK;

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= MAX(1,MIN(M, N)*(MIN(M, N)+4)) for the paths (see comments inside the code): - PATH 1 (M much larger than N) - PATH 1t (N much larger than M) LWORK >= MAX(1,MIN(M, N)*2+MAX(M, N)) for the other paths. For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (12*MIN(M, N))

If INFO = 0, the first NS elements of IWORK are zero. If INFO > 0, then IWORK contains the indices of the eigenvectors that failed to converge in SBDSVDX/SSTEVX.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, then i eigenvectors failed to converge in SBDSVDX/SSTEVX. if INFO = N*2 + 1, an internal error occurred in SBDSVDX

Related Information

For this routine in other precisions, please see *cgesvdx*, *dgesvdx* and *zgesvdx*. It also exists with a native C interface as *LAPACKE_sgesvdx*.

4.10.38 sgesvj

sgesvj computes the singular value decomposition (SVD) of a real M-by-N matrix A, where M >= N. The SVD of A is written as

```
                        [++]   [xx]   [x0]   [xx]
A = U * SIGMA * V^t,    [++] = [xx] * [ox] * [xx]
                        [++]   [xx]
```

where SIGMA is an N-by-N diagonal matrix, U is an M-by-N orthonormal matrix, and V is an N-by-N orthogonal matrix. The diagonal elements of SIGMA are the singular values of A. The columns of U and V are the left and the right singular vectors of A, respectively. sgesvj can sometimes compute tiny singular values and their singular vectors much more accurately than other SVD routines, see below under Further Details.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgesvj(JOBA, JOBU, JOBV, M, N, A, LDA, SVA, MV, V, LDV, WORK,
                  LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgesvj_(const char *joba, const char *jobu, const char *jobv,
             const armpl_int_t *m, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, float *sva, const armpl_int_t *mv,
             float *v, const armpl_int_t *ldv, float *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

JOBA Input parameter.

JOBA is CHARACTER*1

Specifies the structure of A. = 'L': The input matrix A is lower triangular; = 'U': The input matrix A is upper triangular; = 'G': The input matrix A is general M-by-N matrix, M >= N.

JOBU Input parameter.

JOBU is CHARACTER*1

Specifies whether to compute the left singular vectors (columns of U): = 'U': The left singular vectors corresponding to the nonzero singular values are computed and returned in the leading columns of A. See more details in the description of A. The default numerical orthogonality threshold is set to approximately TOL=CTOL*EPS, CTOL=SQRT(M), EPS=SLAMCH('E'). = 'C': Analogous to JOBU='U', except that user can control the level of numerical orthogonality of the computed left singular vectors. TOL can be set to TOL = CTOL*EPS, where CTOL is given on input in the array WORK. No CTOL smaller than ONE is allowed. CTOL greater than 1 / EPS is meaningless. The option 'C' can be used if M*EPS is satisfactory orthogonality of the computed left singular vectors, so CTOL=M could save few sweeps of Jacobi rotations. See the descriptions of A and WORK(1). = 'N': The matrix U is not computed. However, see the description of A.

JOBV Input parameter.

JOBV is CHARACTER*1

Specifies whether to compute the right singular vectors, that is, the matrix V: = 'V' : the matrix V is computed and returned in the array V = 'A' : the Jacobi rotations are applied to the MV-by-N array V. In other words, the right singular vector matrix V is not computed explicitly; instead it is applied to an MV-by-N matrix initially stored in the first MV rows of V. = 'N' : the matrix V is not computed and the array V is not referenced

M Input parameter.

M is INTEGER

The number of rows of the input matrix A. 1/SLAMCH('E') > M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the input matrix A. M >= N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, If JOBU .EQ. 'U' .OR. JOBU .EQ. 'C': If INFO .EQ. 0 : RANKA orthonormal columns of U are returned in the leading RANKA columns of the array A. Here RANKA <= N is the number of computed singular values of A that are above the underflow threshold SLAMCH('S'). The singular vectors corresponding to underflowed or zero singular values are not computed. The value of RANKA is returned in the array WORK as RANKA=NINT(WORK(2)). Also see the descriptions of SVA and WORK. The computed columns of U are mutually numerically orthogonal up to approximately TOL=SQRT(M)*EPS (default); or TOL=CTOL*EPS (JOBU.EQ.'C'), see the description of JOBU. If INFO .GT. 0, the procedure SGESVJ did not converge in the given number of iterations (sweeps).

In that case, the computed columns of U may not be orthogonal up to TOL. The output U (stored in A), SIGMA (given by the computed singular values in SVA(1:N)) and V is still a decomposition of the input matrix A in the sense that the residual ||A-SCALE*U*SIGMA*V^T||_2 / ||A||_2 is small. If JOBU .EQ. 'N': If INFO .EQ. 0 : Note that the left singular vectors are 'for free' in the one-sided Jacobi SVD algorithm. However, if only the singular values are needed, the level of numerical orthogonality of U is not an issue and iterations are stopped when the columns of the iterated matrix are numerically orthogonal up to approximately M*EPS. Thus, on exit, A contains the columns of U scaled with the corresponding singular values. If INFO .GT. 0 : the procedure SGESVJ did not converge in the given number of iterations (sweeps).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

SVA Output parameter.

SVA is REAL

SVA is an array, dimension (N). On exit, If INFO .EQ. 0 : depending on the value SCALE = WORK(1), we have: If SCALE .EQ. ONE: SVA(1:N) contains the computed singular values of A. During the computation SVA contains the Euclidean column norms of the iterated matrices in the array A. If SCALE .NE. ONE: The singular values of A are SCALE*SVA(1:N), and this factored representation is due to the fact that some of the singular values of A might underflow or overflow.

If INFO .GT. 0 : the procedure SGESVJ did not converge in the given number of iterations (sweeps) and SCALE*SVA(1:N) may not be accurate.

MV Input parameter.

MV is INTEGER

If JOBV .EQ. 'A', then the product of Jacobi rotations in SGESVJ is applied to the first MV rows of V. See the description of JOBV.

V Input and output parameter.

V is REAL

V is an array, dimension (LDV, N). If JOBV = 'V', then V contains on exit the N-by-N matrix of the right singular vectors; If JOBV = 'A', then V contains the product of the computed right singular vector matrix and the initial matrix in the array V. If JOBV = 'N', then V is not referenced.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V, LDV .GE. 1. If JOBV .EQ. 'V', then LDV .GE. max(1, N). If JOBV .EQ. 'A', then LDV .GE. max(1, MV) .

WORK Input and output parameter.

WORK is REAL

WORK is an array, dimension (LWORK). On entry, If JOBU .EQ. 'C' : WORK(1) = CTOL, where CTOL defines the threshold for convergence. The process stops if all columns of A are mutually orthogonal up to CTOL*EPS, EPS=SLAMCH('E'). It is required that CTOL >= ONE, i.e. it is not allowed to force the routine to obtain orthogonality below EPSILON. On exit, WORK(1) = SCALE is the scaling factor such that SCALE*SVA(1:N) are the computed singular vcalues of A. (See description of SVA().) WORK(2) = NINT(WORK(2)) is the number of the computed nonzero singular values. WORK(3) = NINT(WORK(3)) is the number of the computed singular values that are larger than the underflow threshold. WORK(4) = NINT(WORK(4)) is the number of sweeps of Jacobi rotations needed for numerical convergence. WORK(5) = max_{i.NE.j} |COS(A(:,i),A(:,j))| in the last sweep. This is useful information in cases when SGESVJ did not converge, as it can be used to estimate whether the output is stil useful and for post festum analysis. WORK(6) = the largest absolute value over all sines of the Jacobi rotation angles in the last sweep. It can be useful for a post festum analysis.

LWORK Input parameter.

LWORK is INTEGER

length of WORK, WORK >= MAX(6,M+N)

INFO Output parameter.

INFO is INTEGER

= 0 : successful exit. < 0 : if INFO = -i, then the i-th argument had an illegal value > 0 : SGESVJ did not converge in the maximal allowed number (30) of sweeps. The output may still be useful. See the description of WORK.

Related Information

For this routine in other precisions, please see *cgesvj*, *dgesvj* and *zgesvj*. It also exists with a native C interface as *LAPACKE_sgesvj*.

4.10.39 sggsvd3

sggsvd3 computes the generalized singular value decomposition (GSVD) of an M-by-N real matrix A and P-by-N real matrix B:

```
U**T*A*Q = D1*( 0 R ),    V**T*B*Q = D2*( 0 R )
```

where U, V and Q are orthogonal matrices. Let K+L = the effective numerical rank of the matrix $(A^T, B^T)^T$, then R is a K+L-by-K+L nonsingular upper triangular matrix, D1 and D2 are M-by-(K+L) and P-by-(K+L) "diagonal" matrices and of the following structures, respectively:

If M-K-L >= 0,

```
                  K  L
     D1 =     K ( I  0 )
              L ( 0  C )
          M-K-L ( 0  0 )

                K  L
     D2 =   L ( 0  S )
          P-L ( 0  0 )

                N-K-L  K    L
( 0 R ) = K (  0   R11  R12 )
          L (  0    0   R22 )
```

where

```
C = diag( ALPHA(K+1), ... , ALPHA(K+L) ),
S = diag( BETA(K+1),  ... , BETA(K+L) ),
C**2 + S**2 = I.

R is stored in A(1:K+L,N-K-L+1:N) on exit.
```

If M-K-L < 0,

```
                  K M-K K+L-M
     D1 =   K ( I  0    0   )
          M-K ( 0  C    0   )

                  K M-K K+L-M
     D2 =   M-K ( 0  S    0  )
          K+L-M ( 0  0    I  )
```

(continues on next page)

(continued from previous page)

```
            P-L ( 0  0    0   )

                 N-K-L  K    M-K  K+L-M
( 0 R ) =      K ( 0    R11  R12  R13  )
             M-K ( 0     0   R22  R23  )
           K+L-M ( 0     0    0   R33  )
```

where

```
C = diag( ALPHA(K+1), ... , ALPHA(M) ),
S = diag( BETA(K+1),  ... , BETA(M) ),
C**2 + S**2 = I.

(R11 R12 R13 ) is stored in A(1:M, N-K-L+1:N), and R33 is stored
( 0  R22 R23 )
in B(M-K+1:L,N+M-K-L+1:N) on exit.
```

The routine computes C, S, R, and optionally the orthogonal transformation matrices U, V and Q.

In particular, if B is an N-by-N nonsingular matrix, then the GSVD of A and B implicitly gives the SVD of A*inv(B):

```
A*inv(B) = U*(D1*inv(D2))*V**T.
```

If (A^T ,B^T)^T has orthonormal columns, then the GSVD of A and B is also equal to the CS decomposition of A and B. Furthermore, the GSVD can be used to derive the solution of the eigenvalue problem:

```
A**T*A x = lambda* B**T*B x.
```

In some literature, the GSVD of A and B is presented in the form

```
U**T*A*X = ( 0 D1 ),   V**T*B*X = ( 0 D2 )
```

where U and V are orthogonal and X is nonsingular, D1 and D2 are ‘’diagonal’‘. The former GSVD form can be converted to the latter form by taking the nonsingular matrix X as

```
X = Q*( I   0    )
      ( 0 inv(R) ).
```

Syntax

Fortran specification:

```
use armpl_library

subroutine sggsvd3(JOBU, JOBV, JOBQ, M, N, P, K, L, A, LDA, B, LDB, ALPHA,
                   BETA, U, LDU, V, LDV, Q, LDQ, WORK, LWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sggsvd3_(const char *jobu, const char *jobv, const char *jobq,
              const armpl_int_t *m, const armpl_int_t *n,
              const armpl_int_t *p, armpl_int_t *k, armpl_int_t *l, float *a,
              const armpl_int_t *lda, float *b, const armpl_int_t *ldb,
              float *alpha, float *beta, float *u, const armpl_int_t *ldu,
              float *v, const armpl_int_t *ldv, float *q,
              const armpl_int_t *ldq, float *work, const armpl_int_t *lwork,
              armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

JOBU Input parameter.

JOBU is CHARACTER*1

= 'U': Orthogonal matrix U is computed; = 'N': U is not computed.

JOBV Input parameter.

JOBV is CHARACTER*1

= 'V': Orthogonal matrix V is computed; = 'N': V is not computed.

JOBQ Input parameter.

JOBQ is CHARACTER*1

= 'Q': Orthogonal matrix Q is computed; = 'N': Q is not computed.

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrices A and B. N >= 0.

P Input parameter.

P is INTEGER

The number of rows of the matrix B. P >= 0.

K Output parameter.

K is INTEGER

L Output parameter.

L is INTEGER

On exit, K and L specify the dimension of the subblocks described in Purpose. K + L = effective numerical rank of $(A^T, B^T)^T$.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, A contains the triangular matrix R, or part of R. See Purpose for details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, N). On entry, the P-by-N matrix B. On exit, B contains the triangular matrix R if M-K-L < 0. See Purpose for details.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, P).

ALPHA Output parameter.

ALPHA is REAL

ALPHA is an array, dimension (N) .

BETA Output parameter.

BETA is REAL

BETA is an array, dimension (N). On exit, ALPHA and BETA contain the generalized singular value pairs of A and B; ALPHA(1:K) = 1, BETA(1:K) = 0, and if M-K-L >= 0, ALPHA(K+1:K+L) = C, BETA(K+1:K+L) = S, or if M-K-L < 0, ALPHA(K+1:M)=C, ALPHA(M+1:K+L)=0 BETA(K+1:M) =S, BETA(M+1:K+L) =1 and ALPHA(K+L+1:N) = 0 BETA(K+L+1:N) = 0

U Output parameter.

U is REAL

U is an array, dimension (LDU, M). If JOBU = 'U', U contains the M-by-M orthogonal matrix U. If JOBU = 'N', U is not referenced.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U. LDU >= max(1, M) if JOBU = 'U'; LDU >= 1 otherwise.

V Output parameter.

V is REAL

V is an array, dimension (LDV, P). If JOBV = 'V', V contains the P-by-P orthogonal matrix V. If JOBV = 'N', V is not referenced.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. LDV >= max(1, P) if JOBV = 'V'; LDV >= 1 otherwise.

Q Output parameter.

Q is REAL

Q is an array, dimension (LDQ, N). If JOBQ = 'Q', Q contains the N-by-N orthogonal matrix Q. If JOBQ = 'N', Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, N) if JOBQ = 'Q'; LDQ >= 1 otherwise.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

On exit, IWORK stores the sorting information. More precisely, the following loop will sort ALPHA for I = K+1, min(M,K+L) swap ALPHA(I) and ALPHA(IWORK(I)) endfor such that ALPHA(1) >= ALPHA(2) >= ... >= ALPHA(N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = 1, the Jacobi-type procedure failed to converge. For further details, see subroutine STGSJA.

Related Information

For this routine in other precisions, please see *cggsvd3*, *dggsvd3* and *zggsvd3*. It also exists with a native C interface as *LAPACKE_sggsvd3*.

4.10.40 sggsvp3

sggsvp3 computes orthogonal matrices U, V and Q such that

```
                   N-K-L  K    L
U**T*A*Q =      K ( 0    A12  A13 )  if M-K-L >= 0;
                L ( 0     0   A23 )
            M-K-L ( 0     0    0  )

                 N-K-L  K    L
       =      K ( 0    A12  A13 )  if M-K-L < 0;
            M-K ( 0     0   A23 )

                 N-K-L  K    L
V**T*B*Q =     L ( 0     0   B13 )
            P-L ( 0     0    0  )
```

where the K-by-K matrix A12 and L-by-L matrix B13 are nonsingular upper triangular; A23 is L-by-L upper triangular if M-K-L >= 0, otherwise A23 is (M-K)-by-L upper trapezoidal. K+L = the effective numerical rank of the (M+P)-by-N matrix $(A^T, B^T)^T$.

This decomposition is the preprocessing step for computing the Generalized Singular Value Decomposition (GSVD), see subroutine SGGSVD3.

Syntax

Fortran specification:

```
use armpl_library

subroutine sggsvp3(JOBU, JOBV, JOBQ, M, P, N, A, LDA, B, LDB, TOLA, TOLB, K,
                   L, U, LDU, V, LDV, Q, LDQ, IWORK, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sggsvp3_(const char *jobu, const char *jobv, const char *jobq,
              const armpl_int_t *m, const armpl_int_t *p,
              const armpl_int_t *n, float *a, const armpl_int_t *lda,
```

(continues on next page)

(continued from previous page)

```
                float *b, const armpl_int_t *ldb, const float *tola,
                const float *tolb, armpl_int_t *k, armpl_int_t *l, float *u,
                const armpl_int_t *ldu, float *v, const armpl_int_t *ldv,
                float *q, const armpl_int_t *ldq, armpl_int_t *iwork,
                float *tau, float *work, const armpl_int_t *lwork,
                armpl_int_t *info, ... );
```

Parameters

JOBU Input parameter.

JOBU is CHARACTER*1

= 'U': Orthogonal matrix U is computed; = 'N': U is not computed.

JOBV Input parameter.

JOBV is CHARACTER*1

= 'V': Orthogonal matrix V is computed; = 'N': V is not computed.

JOBQ Input parameter.

JOBQ is CHARACTER*1

= 'Q': Orthogonal matrix Q is computed; = 'N': Q is not computed.

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

P Input parameter.

P is INTEGER

The number of rows of the matrix B. P >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrices A and B. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, A contains the triangular (or trapezoidal) matrix described in the Purpose section.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, N). On entry, the P-by-N matrix B. On exit, B contains the triangular matrix described in the Purpose section.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, P).

TOLA Input parameter.

TOLA is REAL

TOLB Input parameter.

TOLB is REAL

TOLA and TOLB are the thresholds to determine the effective numerical rank of matrix B and a sub-block of A. Generally, they are set to TOLA = MAX(M, N)*norm(A)*MACHEPS, TOLB = MAX(P, N)*norm(B)*MACHEPS. The size of TOLA and TOLB may affect the size of backward errors of the decomposition.

K Output parameter.

K is INTEGER

L Output parameter.

L is INTEGER

On exit, K and L specify the dimension of the subblocks described in Purpose section. K + L = effective numerical rank of $(A^T, B^T)^T$.

U Output parameter.

U is REAL

U is an array, dimension (LDU, M). If JOBU = 'U', U contains the orthogonal matrix U. If JOBU = 'N', U is not referenced.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U. LDU >= max(1, M) if JOBU = 'U'; LDU >= 1 otherwise.

V Output parameter.

V is REAL

V is an array, dimension (LDV, P). If JOBV = 'V', V contains the orthogonal matrix V. If JOBV = 'N', V is not referenced.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. LDV >= max(1, P) if JOBV = 'V'; LDV >= 1 otherwise.

Q Output parameter.

Q is REAL

Q is an array, dimension (LDQ, N). If JOBQ = 'Q', Q contains the orthogonal matrix Q. If JOBQ = 'N', Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, N) if JOBQ = 'Q'; LDQ >= 1 otherwise.

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

TAU Output parameter.

TAU is REAL

TAU is an array, dimension (N) .

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cggsvp3*, *dggsvp3* and *zggsvp3*. It also exists with a native C interface as *LAPACKE_sggsvp3*.

4.10.41 sorgbr

sorgbr generates one of the real orthogonal matrices Q or P^T determined by SGEBRD when reducing a real matrix A to bidiagonal form: A = Q * B * P^T . Q and P^T are defined as products of elementary reflectors H(i) or G(i) respectively.

If VECT = 'Q', A is assumed to have been an M-by-K matrix, and Q is of order M: if m >= k, Q = H(1) H(2) . . . H(k) and sorgbr returns the first n columns of Q, where m >= n >= k; if m < k, Q = H(1) H(2) . . . H(m-1) and sorgbr returns Q as an M-by-M matrix.

If VECT = 'P', A is assumed to have been a K-by-N matrix, and P^T is of order N: if k < n, P^T = G(k) . . . G(2) G(1) and sorgbr returns the first m rows of P^T , where n >= m >= k; if k >= n, P^T = G(n-1) . . . G(2) G(1) and sorgbr returns P^T as an N-by-N matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine sorgbr(VECT, M, N, K, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sorgbr_(const char *vect, const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *k, float *a, const armpl_int_t *lda,
             const float *tau, float *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

VECT Input parameter.

VECT is CHARACTER*1

Specifies whether the matrix Q or the matrix P^T is required, as defined in the transformation applied by SGEBRD: = 'Q': generate Q; = 'P': generate P^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix Q or P^T to be returned. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix Q or P^T to be returned. N >= 0. If VECT = 'Q', M >= N >= min(M, K); if VECT = 'P', N >= M >= min(N, K).

K Input parameter.

K is INTEGER

If VECT = 'Q', the number of columns in the original M-by-K matrix reduced by SGEBRD. If VECT = 'P', the number of rows in the original K-by-N matrix reduced by SGEBRD. K >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the vectors which define the elementary reflectors, as returned by SGEBRD. On exit, the M-by-N matrix Q or P^T .

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Input parameter.

TAU is REAL

TAU is an array, dimension. (min(M, K)) if VECT = 'Q' (min(N, K)) if VECT = 'P' TAU(i) must contain the scalar factor of the elementary reflector H(i) or G(i), which determines Q or P^T , as returned by SGEBRD in its array argument TAUQ or TAUP.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,min(M, N)). For optimum performance LWORK >= min(M, N)*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dorgbr*. It also exists with a native C interface as *LAPACKE_sorgbr*.

4.10.42 sormbr

If VECT = 'Q', `sormbr` overwrites the general real M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'T': Q^T * C C * Q^T

If VECT = 'P', `sormbr` overwrites the general real M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': P * C C * P TRANS = 'T': P^T * C C * P^T

Here Q and P^T are the orthogonal matrices determined by SGEBRD when reducing a real matrix A to bidiagonal form: A = Q * B * P^T . Q and P^T are defined as products of elementary reflectors H(i) and G(i) respectively.

Let nq = m if SIDE = 'L' and nq = n if SIDE = 'R'. Thus nq is the order of the orthogonal matrix Q or P^T that is applied.

If VECT = 'Q', A is assumed to have been an NQ-by-K matrix: if nq >= k, Q = H(1) H(2) . . . H(k); if nq < k, Q = H(1) H(2) . . . H(nq-1).

If VECT = 'P', A is assumed to have been a K-by-NQ matrix: if k < nq, P = G(1) G(2) . . . G(k); if k >= nq, P = G(1) G(2) . . . G(nq-1).

Syntax

Fortran specification:

```
use armpl_library

subroutine sormbr(VECT, SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK,
                  LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sormbr_(const char *vect, const char *side, const char *trans,
             const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *k,
             const float *a, const armpl_int_t *lda, const float *tau,
             float *c, const armpl_int_t *ldc, float *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

VECT Input parameter.

VECT is CHARACTER*1

= 'Q': apply Q or Q^T ; = 'P': apply P or P^T .

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q, Q^T , P or P^T from the Left; = 'R': apply Q, Q^T , P or P^T from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q or P; = 'T': Transpose, apply Q^T or P^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

If VECT = 'Q', the number of columns in the original matrix reduced by SGEBRD. If VECT = 'P', the number of rows in the original matrix reduced by SGEBRD. K >= 0.

A Input parameter.

A is REAL

A is an array, dimension. (LDA,min(nq, K)) if VECT = 'Q' (LDA,nq) if VECT = 'P' The vectors which define the elementary reflectors H(i) and G(i), whose products determine the matrices Q and P, as returned by SGEBRD.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If VECT = 'Q', LDA >= max(1,nq); if VECT = 'P', LDA >= max(1,min(nq, K)).

TAU Input parameter.

TAU is REAL

TAU is an array, dimension (min(nq, K)). TAU(i) must contain the scalar factor of the elementary reflector H(i) or G(i) which determines Q or P, as returned by SGEBRD in the array argument TAUQ or TAUP.

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q or P* C or P^T * C or C*P or C*P^T .

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, N); if SIDE = 'R', LWORK >= max(1, M). For optimum performance LWORK >= N*NB if SIDE = 'L', and LWORK >= M*NB if SIDE = 'R', where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dormbr*. It also exists with a native C interface as *LAPACKE_sormbr*.

4.10.43 stgsja

stgsja computes the generalized singular value decomposition (GSVD) of two real upper triangular (or trapezoidal) matrices A and B.

On entry, it is assumed that matrices A and B have the following forms, which may be obtained by the preprocessing subroutine SGGSVP from a general M-by-N matrix A and P-by-N matrix B:

```
           N-K-L  K    L
A =    K ( 0    A12  A13 ) if M-K-L >= 0;
       L ( 0     0   A23 )
   M-K-L ( 0     0    0  )

         N-K-L  K    L
A =  K ( 0    A12  A13 ) if M-K-L < 0;
   M-K ( 0     0   A23 )

         N-K-L  K    L
B =  L ( 0     0   B13 )
   P-L ( 0     0    0  )
```

where the K-by-K matrix A12 and L-by-L matrix B13 are nonsingular upper triangular; A23 is L-by-L upper triangular if M-K-L >= 0, otherwise A23 is (M-K)-by-L upper trapezoidal.

On exit,

```
U**T *A*Q = D1*( 0 R ),    V**T *B*Q = D2*( 0 R ),
```

where U, V and Q are orthogonal matrices. R is a nonsingular upper triangular matrix, and D1 and D2 are ‘’diagonal” matrices, which are of the following structures:

If M-K-L >= 0,

```
                  K  L
     D1 =     K ( I  0 )
              L ( 0  C )
            M-K-L ( 0  0 )

                  K  L
     D2 = L   ( 0  S )
            P-L ( 0  0 )

               N-K-L  K    L
( 0 R ) = K (  0   R11  R12 ) K
          L (  0    0   R22 ) L
```

where

```
C = diag( ALPHA(K+1), ... , ALPHA(K+L) ),
S = diag( BETA(K+1),  ... , BETA(K+L) ),
C**2 + S**2 = I.

R is stored in A(1:K+L,N-K-L+1:N) on exit.
```

If M-K-L < 0,

```
            K M-K K+L-M
D1 =   K ( I  0    0   )
     M-K ( 0  C    0   )

              K M-K K+L-M
D2 =   M-K ( 0  S    0   )
     K+L-M ( 0  0    I   )
       P-L ( 0  0    0   )

            N-K-L  K   M-K  K+L-M
```

(0 R) = K (0 R11 R12 R13)

```
  M-K ( 0     0   R22  R23  )
K+L-M ( 0     0    0   R33  )
```

where C = diag(ALPHA(K+1), ... , ALPHA(M)), S = diag(BETA(K+1), ... , BETA(M)), C**2 + S**2 = I.

R = (R11 R12 R13) is stored in A(1:M, N-K-L+1:N) and R33 is stored

```
(  0  R22 R23 )
```

in B(M-K+1:L,N+M-K-L+1:N) on exit.

The computation of the orthogonal transformation matrices U, V or Q is optional. These matrices may either be formed explicitly, or they may be postmultiplied into input matrices U1, V1, or Q1.

Syntax

Fortran specification:

```
use armpl_library

subroutine stgsja(JOBU, JOBV, JOBQ, M, P, N, K, L, A, LDA, B, LDB, TOLA, TOLB,
                  ALPHA, BETA, U, LDU, V, LDV, Q, LDQ, WORK, NCYCLE, INFO)
```

C specification:

```
#include "armpl.h"

void stgsja_(const char *jobu, const char *jobv, const char *jobq,
             const armpl_int_t *m, const armpl_int_t *p, const armpl_int_t *n,
             const armpl_int_t *k, const armpl_int_t *l, float *a,
             const armpl_int_t *lda, float *b, const armpl_int_t *ldb,
             const float *tola, const float *tolb, float *alpha, float *beta,
             float *u, const armpl_int_t *ldu, float *v,
             const armpl_int_t *ldv, float *q, const armpl_int_t *ldq,
             float *work, armpl_int_t *ncycle, armpl_int_t *info, ... );
```

Parameters

JOBU Input parameter.

JOBU is CHARACTER*1

= 'U': U must contain an orthogonal matrix U1 on entry, and the product U1* U is returned; = 'I': U is initialized to the unit matrix, and the orthogonal matrix U is returned; = 'N': U is not computed.

JOBV Input parameter.

JOBV is CHARACTER*1

= 'V': V must contain an orthogonal matrix V1 on entry, and the product V1* V is returned; = 'I': V is initialized to the unit matrix, and the orthogonal matrix V is returned; = 'N': V is not computed.

JOBQ Input parameter.

JOBQ is CHARACTER*1

= 'Q': Q must contain an orthogonal matrix Q1 on entry, and the product Q1* Q is returned; = 'I': Q is initialized to the unit matrix, and the orthogonal matrix Q is returned; = 'N': Q is not computed.

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

P Input parameter.

P is INTEGER

The number of rows of the matrix B. P >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrices A and B. N >= 0.

K Input parameter.

K is INTEGER

L Input parameter.

L is INTEGER

K and L specify the subblocks in the input matrices A and B: A23 = A(K+1:MIN(K+L, M),N-L+1:N) and B13 = B(1:L,N-L+1:N) of A and B, whose GSVD is going to be computed by STGSJA. See Further Details.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, A(N-K+1:N,1:MIN(K+L, M)) contains the triangular matrix R or part of R. See Purpose for details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, N). On entry, the P-by-N matrix B. On exit, if necessary, B(M-K+1:L,N+M-K-L+1:N) contains a part of R. See Purpose for details.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, P).

TOLA Input parameter.

TOLA is REAL

TOLB Input parameter.

TOLB is REAL

TOLA and TOLB are the convergence criteria for the Jacobi- Kogbetliantz iteration procedure. Generally, they are the same as used in the preprocessing step, say TOLA = max(M, N)*norm(A)*MACHEPS, TOLB = max(P, N)*norm(B)*MACHEPS.

ALPHA Output parameter.

ALPHA is REAL

ALPHA is an array, dimension (N) .

BETA Output parameter.

BETA is REAL

BETA is an array, dimension (N). On exit, ALPHA and BETA contain the generalized singular value pairs of A and B; ALPHA(1:K) = 1, BETA(1:K) = 0, and if M-K-L >= 0, ALPHA(K+1:K+L) = diag(C), BETA(K+1:K+L) = diag(S), or if M-K-L < 0, ALPHA(K+1:M)= C, ALPHA(M+1:K+L)= 0 BETA(K+1:M) = S, BETA(M+1:K+L) = 1. Furthermore, if K+L < N, ALPHA(K+L+1:N) = 0 and BETA(K+L+1:N) = 0.

U Input and output parameter.

U is REAL

U is an array, dimension (LDU, M). On entry, if JOBU = 'U', U must contain a matrix U1 (usually the orthogonal matrix returned by SGGSVP). On exit, if JOBU = 'I', U contains the orthogonal matrix U; if JOBU = 'U', U contains the product U1*U. If JOBU = 'N', U is not referenced.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U. LDU >= max(1, M) if JOBU = 'U'; LDU >= 1 otherwise.

V Input and output parameter.

V is REAL

V is an array, dimension (LDV, P). On entry, if JOBV = 'V', V must contain a matrix V1 (usually the orthogonal matrix returned by SGGSVP). On exit, if JOBV = 'I', V contains the orthogonal matrix V; if JOBV = 'V', V contains the product V1*V. If JOBV = 'N', V is not referenced.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. LDV >= max(1, P) if JOBV = 'V'; LDV >= 1 otherwise.

Q Input and output parameter.

Q is REAL

Q is an array, dimension (LDQ, N). On entry, if JOBQ = 'Q', Q must contain a matrix Q1 (usually the orthogonal matrix returned by SGGSVP). On exit, if JOBQ = 'I', Q contains the orthogonal matrix Q; if JOBQ = 'Q', Q contains the product Q1*Q. If JOBQ = 'N', Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, N) if JOBQ = 'Q'; LDQ >= 1 otherwise.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (2*N) .

NCYCLE Output parameter.

NCYCLE is INTEGER

The number of cycles required for convergence.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. = 1: the procedure does not converge after MAXIT cycles.

Related Information

For this routine in other precisions, please see *ctgsja*, *dtgsja* and *ztgsja*. It also exists with a native C interface as *LAPACKE_stgsja*.

4.10.44 zbdsqr

zbdsqr computes the singular values and, optionally, the right and/or left singular vectors from the singular value decomposition (SVD) of a real N-by-N (upper or lower) bidiagonal matrix B using the implicit zero-shift QR algorithm. The SVD of B has the form

```
B = Q * S * P**H
```

where S is the diagonal matrix of singular values, Q is an orthogonal matrix of left singular vectors, and P is an orthogonal matrix of right singular vectors. If left singular vectors are requested, this subroutine actually returns U*Q instead of Q, and, if right singular vectors are requested, this subroutine returns P^H *VT instead of P^H , for given complex input matrices U and VT. When U and VT are the unitary matrices that reduce a general matrix A to bidiagonal form: A = U*B*VT, as computed by ZGEBRD, then

```
A = (U*Q) * S * (P**H*VT)
```

is the SVD of A. Optionally, the subroutine may also compute Q^H *C for a given complex input matrix C.

See "Computing Small Singular Values of Bidiagonal Matrices With Guaranteed High Relative Accuracy," by J. Demmel and W. Kahan, LAPACK Working Note #3 (or SIAM J. Sci. Statist. Comput. vol. 11, no. 5, pp. 873-912, Sept 1990) and "Accurate singular values and differential qd algorithms," by B. Parlett and V. Fernando, Technical Report CPAM-554, Mathematics Department, University of California at Berkeley, July 1992 for a detailed description of the algorithm.

Syntax

Fortran specification:

```
use armpl_library

subroutine zbdsqr(UPLO, N, NCVT, NRU, NCC, D, E, VT, LDVT, U, LDU, C, LDC,
                  RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zbdsqr_(const char *uplo, const armpl_int_t *n, const armpl_int_t *ncvt,
             const armpl_int_t *nru, const armpl_int_t *ncc, double *d,
             double *e, armpl_doublecomplex_t *vt, const armpl_int_t *ldvt,
             armpl_doublecomplex_t *u, const armpl_int_t *ldu,
             armpl_doublecomplex_t *c, const armpl_int_t *ldc, double *rwork,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': B is upper bidiagonal; = 'L': B is lower bidiagonal.

N Input parameter.

N is INTEGER

The order of the matrix B. N >= 0.

NCVT Input parameter.

NCVT is INTEGER

The number of columns of the matrix VT. NCVT >= 0.

NRU Input parameter.

NRU is INTEGER

The number of rows of the matrix U. NRU >= 0.

NCC Input parameter.

NCC is INTEGER

The number of columns of the matrix C. NCC >= 0.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry, the n diagonal elements of the bidiagonal matrix B. On exit, if INFO=0, the singular values of B in decreasing order.

E Input and output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N-1). On entry, the N-1 offdiagonal elements of the bidiagonal matrix B. On exit, if INFO = 0, E is destroyed; if INFO > 0, D and E will contain the diagonal and superdiagonal elements of a bidiagonal matrix orthogonally equivalent to the one given as input.

VT Input and output parameter.

VT is COMPLEX*16

VT is an array, dimension (LDVT, NCVT). On entry, an N-by-NCVT matrix VT. On exit, VT is overwritten by P^H * VT. Not referenced if NCVT = 0.

LDVT Input parameter.

LDVT is INTEGER

The leading dimension of the array VT. LDVT >= max(1, N) if NCVT > 0; LDVT >= 1 if NCVT = 0.

U Input and output parameter.

U is COMPLEX*16

U is an array, dimension (LDU, N). On entry, an NRU-by-N matrix U. On exit, U is overwritten by U * Q. Not referenced if NRU = 0.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U. LDU >= max(1, NRU).

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, NCC). On entry, an N-by-NCC matrix C. On exit, C is overwritten by Q^H * C. Not referenced if NCC = 0.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, N) if NCC > 0; LDC >=1 if NCC = 0.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (4*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: If INFO = -i, the i-th argument had an illegal value > 0: the algorithm did not converge; D and E contain the elements of a bidiagonal matrix which is orthogonally similar to the input matrix B; if INFO = i, i elements of E have not converged to zero.

Related Information

For this routine in other precisions, please see *cbdsqr*, *dbdsqr* and *sbdsqr*. It also exists with a native C interface as *LAPACKE_zbdsqr*.

4.10.45 zgbbrd

zgbbrd reduces a complex general m-by-n band matrix A to real upper bidiagonal form B by a unitary transformation: Q^H * A * P = B.

The routine computes B, and optionally forms Q or P^H , or computes Q^H *C for a given matrix C.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgbbrd(VECT, M, N, NCC, KL, KU, AB, LDAB, D, E, Q, LDQ, PT, LDPT,
                  C, LDC, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgbbrd_(const char *vect, const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *ncc, const armpl_int_t *kl,
             const armpl_int_t *ku, armpl_doublecomplex_t *ab,
             const armpl_int_t *ldab, double *d, double *e,
             armpl_doublecomplex_t *q, const armpl_int_t *ldq,
             armpl_doublecomplex_t *pt, const armpl_int_t *ldpt,
             armpl_doublecomplex_t *c, const armpl_int_t *ldc,
             armpl_doublecomplex_t *work, double *rwork, armpl_int_t *info,
             ... );
```

Parameters

VECT Input parameter.

VECT is CHARACTER*1

Specifies whether or not the matrices Q and P^H are to be formed. = 'N': do not form Q or P^H ; = 'Q': form Q only; = 'P': form P^H only; = 'B': form both.

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

NCC Input parameter.

NCC is INTEGER

The number of columns of the matrix C. NCC >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals of the matrix A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals of the matrix A. KU >= 0.

AB Input and output parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). On entry, the m-by-n band matrix A, stored in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(ku+1+i-j,j) = A(i,j) for max(1,j-ku)<=i<=min(m,j+kl). On exit, A is overwritten by values generated during the reduction.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array A. LDAB >= KL+KU+1.

D Output parameter.

D is DOUBLE PRECISION

D is an array, dimension (min(M, N)). The diagonal elements of the bidiagonal matrix B.

E Output parameter.

E is DOUBLE PRECISION

E is an array, dimension (min(M, N)-1). The superdiagonal elements of the bidiagonal matrix B.

Q Output parameter.

Q is COMPLEX*16

Q is an array, dimension (LDQ, M). If VECT = 'Q' or 'B', the m-by-m unitary matrix Q. If VECT = 'N' or 'P', the array Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, M) if VECT = 'Q' or 'B'; LDQ >= 1 otherwise.

PT Output parameter.

PT is COMPLEX*16

PT is an array, dimension (LDPT, N). If VECT = 'P' or 'B', the n-by-n unitary matrix P'. If VECT = 'N' or 'Q', the array PT is not referenced.

LDPT Input parameter.

LDPT is INTEGER

The leading dimension of the array PT. LDPT >= max(1, N) if VECT = 'P' or 'B'; LDPT >= 1 otherwise.

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, NCC). On entry, an m-by-ncc matrix C. On exit, C is overwritten by Q^H *C. C is not referenced if NCC = 0.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M) if NCC > 0; LDC >= 1 if NCC = 0.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (max(M, N)) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (max(M, N)) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cgbbrd*, *dgbbrd* and *sgbbrd*. It also exists with a native C interface as *LAPACKE_zgbbrd*.

4.10.46 zgebrd

zgebrd reduces a general complex M-by-N matrix A to upper or lower bidiagonal form B by a unitary transformation: Q^H * A * P = B.

If m >= n, B is upper bidiagonal; if m < n, B is lower bidiagonal.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgebrd(M, N, A, LDA, D, E, TAUQ, TAUP, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgebrd_(const armpl_int_t *m, const armpl_int_t *n,
             armpl_doublecomplex_t *a, const armpl_int_t *lda, double *d,
             double *e, armpl_doublecomplex_t *tauq,
             armpl_doublecomplex_t *taup, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows in the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns in the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the M-by-N general matrix to be reduced. On exit, if m >= n, the diagonal and the first superdiagonal are overwritten with the upper bidiagonal matrix B; the elements below the diagonal, with the array TAUQ, represent the unitary matrix Q as a product of elementary reflectors, and the elements above the first superdiagonal, with the array TAUP, represent the unitary matrix P as a product of elementary reflectors; if m < n, the diagonal and the first subdiagonal are overwritten with the lower bidiagonal matrix B; the elements below the first subdiagonal, with the array TAUQ, represent the unitary matrix Q as a product of elementary reflectors, and the elements above the diagonal, with the array TAUP, represent the unitary matrix P as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

D Output parameter.

D is DOUBLE PRECISION

D is an array, dimension (min(M, N)). The diagonal elements of the bidiagonal matrix B: D(i) = A(i,i).

E Output parameter.

E is DOUBLE PRECISION

E is an array, dimension (min(M, N)-1). The off-diagonal elements of the bidiagonal matrix B: if m >= n, E(i) = A(i,i+1) for i = 1,2,...,n-1; if m < n, E(i) = A(i+1,i) for i = 1,2,...,m-1.

TAUQ Output parameter.

TAUQ is COMPLEX*16

TAUQ is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors which represent the unitary matrix Q. See Further Details.

TAUP Output parameter.

TAUP is COMPLEX*16

TAUP is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors which represent the unitary matrix P. See Further Details.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= max(1, M, N). For optimum performance LWORK >= (M+N)*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cgebrd*, *dgebrd* and *sgebrd*. It also exists with a native C interface as *LAPACKE_zgebrd*.

4.10.47 zgejsv

zgejsv computes the singular value decomposition (SVD) of a complex M-by-N matrix [A], where M >= N. The SVD of [A] is written as

```
[A] = [U] * [SIGMA] * [V]^*,
```

where [SIGMA] is an N-by-N (M-by-N) matrix which is zero except for its N diagonal elements, [U] is an M-by-N (or M-by-M) unitary matrix, and [V] is an N-by-N unitary matrix. The diagonal elements of [SIGMA] are the singular values of [A]. The columns of [U] and [V] are the left and the right singular vectors of [A], respectively. The matrices [U] and [V] are computed and stored in the arrays U and V, respectively. The diagonal of [SIGMA] is computed and stored in the array SVA.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgejsv(JOBA, JOBU, JOBV, JOBR, JOBT, JOBP, M, N, A, LDA, SVA, U,
                  LDU, V, LDV, CWORK, LWORK, RWORK, LRWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgejsv_(const char *joba, const char *jobu, const char *jobv,
             const char *jobr, const char *jobt, const char *jobp,
             const armpl_int_t *m, const armpl_int_t *n,
             armpl_doublecomplex_t *a, const armpl_int_t *lda, double *sva,
             armpl_doublecomplex_t *u, const armpl_int_t *ldu,
             armpl_doublecomplex_t *v, const armpl_int_t *ldv,
             armpl_doublecomplex_t *cwork, const armpl_int_t *lwork,
             double *rwork, const armpl_int_t *lrwork, armpl_int_t *iwork,
             armpl_int_t *info, ... );
```

Parameters

JOBA Input parameter.

JOBA is CHARACTER*1

Specifies the level of accuracy: = 'C': This option works well (high relative accuracy) if A = B * D, with well-conditioned B and arbitrary diagonal matrix D. The accuracy cannot be spoiled by COLUMN scaling. The accuracy of the computed output depends on the condition of B, and the procedure aims at the best theoretical accuracy. The relative error max_{i=1:N}|d sigma_i| / sigma_i is bounded by f(M, N)*epsilon* cond(B), independent of D. The input matrix is preprocessed with the QRF with column pivoting. This initial preprocessing and preconditioning by a rank revealing QR factorization is common for all values of JOBA. Additional actions are specified as follows: = 'E': Computation as with 'C' with an additional estimate of the condition number of B. It provides a realistic error bound. = 'F': If A = D1 * C * D2 with ill-conditioned diagonal scalings D1, D2, and well-conditioned matrix C, this option gives higher accuracy than the 'C' option. If the structure of the input matrix is not known, and relative accuracy is desirable, then this option is advisable. The input matrix A is preprocessed with QR factorization with FULL (row and column) pivoting. = 'G' Computation as with 'F' with an additional estimate of the condition number of B, where A=B*D. If A has heavily weighted rows, then using this condition number gives too pessimistic error bound. = 'A': Small singular values are not well determined by the data and are considered as noisy; the matrix is treated as numerically rank defficient. The error in the computed singular values is bounded by f(m,n)*epsilon*||A||. The computed SVD A = U * S * V^* restores A up to f(m,n)*epsilon*||A||. This gives the procedure the licence to discard (set to zero) all singular values below N*epsilon*||A||. = 'R': Similar as in 'A'. Rank revealing property of the initial QR factorization is used do reveal (using triangular factor) a gap sigma_{r+1} < epsilon * sigma_r in which case the numerical RANK is declared to be r. The SVD is computed with absolute error bounds, but more accurately than with 'A'.

JOBU Input parameter.

JOBU is CHARACTER*1

Specifies whether to compute the columns of U: = 'U': N columns of U are returned in the array U. = 'F': full set of M left sing. vectors is returned in the array U. = 'W': U may be used as workspace of length M*N. See the description of U. = 'N': U is not computed.

JOBV Input parameter.

JOBV is CHARACTER*1

Specifies whether to compute the matrix V: = 'V': N columns of V are returned in the array V; Jacobi rotations are not explicitly accumulated. = 'J': N columns of V are returned in the array V, but they are computed as the product of Jacobi rotations, if JOBT .EQ. 'N'. = 'W': V may be used as workspace of length N*N. See the description of V. = 'N': V is not computed.

JOBR Input parameter.

JOBR is CHARACTER*1

Specifies the RANGE for the singular values. Issues the licence to set to zero small positive singular values if they are outside specified range. If A .NE. 0 is scaled so that the largest singular value of c* A is around SQRT(BIG), BIG=DLAMCH('O'), then JOBR issues the licence to kill columns of A

whose norm in c* A is less than SQRT(SFMIN) (for JOBR.EQ.'R'), or less than SMALL=SFMIN/EPSLN, where SFMIN=DLAMCH('S'), EPSLN=DLAMCH('E'). = 'N': Do not kill small columns of c*A. This option assumes that BLAS and QR factorizations and triangular solvers are implemented to work in that range. If the condition of A is greater than BIG, use ZGESVJ. = 'R': RESTRICTED range for sigma(c*A) is [SQRT(SFMIN), SQRT(BIG)] (roughly, as described above). This option is recommended. ========== For computing the singular values in the FULL range [SFMIN,BIG] use ZGESVJ.

JOBT Input parameter.

JOBT is CHARACTER*1

If the matrix is square then the procedure may determine to use transposed A if A^* seems to be better with respect to convergence. If the matrix is not square, JOBT is ignored. The decision is based on two values of entropy over the adjoint orbit of A^* * A. See the descriptions of WORK(6) and WORK(7). = 'T': transpose if entropy test indicates possibly faster convergence of Jacobi process if A^* is taken as input. If A is replaced with A^*, then the row pivoting is included automatically. = 'N': do not speculate. The option 'T' can be used to compute only the singular values, or the full SVD (U, SIGMA and V). For only one set of singular vectors (U or V), the caller should provide both U and V, as one of the matrices is used as workspace if the matrix A is transposed. The implementer can easily remove this constraint and make the code more complicated. See the descriptions of U and V. In general, this option is considered experimental, and 'N'; should be preferred. This is subject to changes in the future.

JOBP Input parameter.

JOBP is CHARACTER*1

Issues the licence to introduce structured perturbations to drown denormalized numbers. This licence should be active if the denormals are poorly implemented, causing slow computation, especially in cases of fast convergence (!). For details see [1,2]. For the sake of simplicity, this perturbations are included only when the full SVD or only the singular values are requested. The implementer/user can easily add the perturbation for the cases of computing one set of singular vectors. = 'P': introduce perturbation = 'N': do not perturb

M Input parameter.

M is INTEGER

The number of rows of the input matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the input matrix A. M >= N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

SVA Output parameter.

SVA is DOUBLE PRECISION

SVA is an array, dimension (N). On exit, - For WORK(1)/WORK(2) = ONE: The singular values of A. During the computation SVA contains Euclidean column norms of the iterated matrices in the array A. - For WORK(1) .NE. WORK(2): The singular values of A are (WORK(1)/WORK(2)) * SVA(1:N). This factored form is used if sigma_max(A) overflows or if small singular values have been saved from underflow by scaling the input matrix A. - If JOBR='R' then some of the singular values may be returned as exact zeros obtained by "set to zero" because they are below the numerical rank threshold or are denormalized numbers.

U Output parameter.

U is COMPLEX*16

U is an array, dimension (LDU, N). If JOBU = 'U', then U contains on exit the M-by-N matrix of the left singular vectors. If JOBU = 'F', then U contains on exit the M-by-M matrix of the left singular vectors, including an ONB of the orthogonal complement of the Range(A). If JOBU = 'W' .AND. (JOBV.EQ.'V' .AND. JOBT.EQ.'T' .AND. M.EQ.N), then U is used as workspace if the procedure replaces A with A^*. In that case, [V] is computed in U as left singular vectors of A^* and then copied back to the V array. This 'W' option is just a reminder to the caller that in this case U is reserved as workspace of length N*N. If JOBU = 'N' U is not referenced, unless JOBT='T'.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U, LDU >= 1. IF JOBU = 'U' or 'F' or 'W', then LDU >= M.

V Output parameter.

V is COMPLEX*16

V is an array, dimension (LDV, N). If JOBV = 'V', 'J' then V contains on exit the N-by-N matrix of the right singular vectors; If JOBV = 'W', AND (JOBU.EQ.'U' AND JOBT.EQ.'T' AND M.EQ.N), then V is used as workspace if the pprocedure replaces A with A^*. In that case, [U] is computed in V as right singular vectors of A^* and then copied back to the U array. This 'W' option is just a reminder to the caller that in this case V is reserved as workspace of length N*N. If JOBV = 'N' V is not referenced, unless JOBT='T'.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V, LDV >= 1. If JOBV = 'V' or 'J' or 'W', then LDV >= N.

CWORK Output parameter.

CWORK is COMPLEX*16

CWORK is an array, dimension (MAX(2, LWORK)). If the call to ZGEJSV is a workspace query (indicated by LWORK=-1 or LRWORK=-1), then on exit CWORK(1) contains the required length of CWORK for the job parameters used in the call.

LWORK Input parameter.

LWORK is INTEGER

Length of CWORK to confirm proper allocation of workspace. LWORK depends on the job:

1. If only SIGMA is needed (JOBU.EQ.'N', JOBV.EQ.'N') and 1.1 .. no scaled condition estimate required (JOBA.NE.'E'.AND.JOBA.NE.'G'): LWORK >= 2*N+1. This is the minimal requirement. ->> For optimal performance (blocked code) the optimal value is LWORK >= N + (N+1)*NB. Here NB is the optimal block size for ZGEQP3 and ZGEQRF. In general, optimal LWORK is computed as LWORK >= max(N+LWORK(ZGEQP3),N+LWORK(ZGEQRF), LWORK(ZGESVJ)). 1.2. .. an estimate of the scaled condition number of A is required (JOBA='E', or 'G'). In this case, LWORK the minimal requirement is LWORK >= N* N + 2*N. ->> For optimal performance (blocked code) the optimal value is LWORK >= max(N+(N+1)*NB, N*N+2*N)=N**2+2*N. In general, the optimal length LWORK is computed as LWORK >= max(N+LWORK(ZGEQP3),N+LWORK(ZGEQRF), LWORK(ZGESVJ), N*N+LWORK(ZPOCON)). 2. If SIGMA and the right singular vectors are needed (JOBV.EQ.'V'), (JOBU.EQ.'N') 2.1 .. no scaled condition estimate requested (JOBE.EQ.'N'): -> the minimal requirement is LWORK >= 3*N. -> For optimal performance, LWORK >= max(N+(N+1)*NB, 2*N+N*NB)=2*N+N*NB, where NB is the optimal block size for ZGEQP3, ZGEQRF, ZGELQ, ZUNMLQ. In general, the optimal length LWORK is computed as LWORK >= max(N+LWORK(ZGEQP3), N+LWORK(ZGESVJ), N+LWORK(ZGELQF), 2*N+LWORK(ZGEQRF), N+LWORK(ZUNMLQ)). 2.2 .. an estimate of the scaled condition number of A is required (JOBA='E', or 'G'). -> the minimal requirement is LWORK >= 3*N. -> For optimal performance, LWORK >= max(N+(N+1)*NB, 2*N,2*N+N*NB)=2*N+N*NB, where NB is the optimal block size for ZGEQP3, ZGEQRF, ZGELQ, ZUNMLQ. In general, the optimal length LWORK is computed as LWORK >= max(N+LWORK(ZGEQP3), LWORK(ZPOCON), N+LWORK(ZGESVJ),

N+LWORK(ZGELQF), 2*N+LWORK(ZGEQRF), N+LWORK(ZUNMLQ)). 3. If SIGMA and the left singular vectors are needed 3.1 .. no scaled condition estimate requested (JOBE.EQ.'N'): -> the minimal requirement is LWORK >= 3*N. -> For optimal performance: if JOBU.EQ.'U' :: LWORK >= max(3*N, N+(N+1)*NB, 2*N+N*NB)=2*N+N*NB, where NB is the optimal block size for ZGEQP3, ZGEQRF, ZUNMQR. In general, the optimal length LWORK is computed as LWORK >= max(N+LWORK(ZGEQP3), 2*N+LWORK(ZGEQRF), N+LWORK(ZUNMQR)). 3.2 .. an estimate of the scaled condition number of A is required (JOBA='E', or 'G'). -> the minimal requirement is LWORK >= 3*N. -> For optimal performance: if JOBU.EQ.'U' :: LWORK >= max(3*N, N+(N+1)*NB, 2*N+N*NB)=2*N+N*NB, where NB is the optimal block size for ZGEQP3, ZGEQRF, ZUNMQR. In general, the optimal length LWORK is computed as LWORK >= max(N+LWORK(ZGEQP3),N+LWORK(ZPOCON), 2*N+LWORK(ZGEQRF), N+LWORK(ZUNMQR)). 4. If the full SVD is needed: (JOBU.EQ.'U' or JOBU.EQ.'F') and 4.1. if JOBV.EQ.'V' the minimal requirement is LWORK >= 5*N+2*N*N. 4.2. if JOBV.EQ.'J' the minimal requirement is LWORK >= 4*N+N*N. In both cases, the allocated CWORK can accommodate blocked runs of ZGEQP3, ZGEQRF, ZGELQF, SUNMQR, ZUNMLQ.

If the call to ZGEJSV is a workspace query (indicated by LWORK=-1 or LRWORK=-1), then on exit CWORK(1) contains the optimal and CWORK(2) contains the minimal length of CWORK for the job parameters used in the call.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (MAX(7, LWORK)). On exit, RWORK(1) = Determines the scaling factor SCALE = RWORK(2) / RWORK(1) such that SCALE*SVA(1:N) are the computed singular values of A. (See the description of SVA().) RWORK(2) = See the description of RWORK(1). RWORK(3) = SCONDA is an estimate for the condition number of column equilibrated A. (If JOBA .EQ. 'E' or 'G') SCONDA is an estimate of SQRT(||(R^* * R)^(-1)||_1). It is computed using SPOCON. It holds N^(-1/4) * SCONDA <= ||R^(-1)||_2 <= N^(1/4) * SCONDA where R is the triangular factor from the QRF of A. However, if R is truncated and the numerical rank is determined to be strictly smaller than N, SCONDA is returned as -1, thus indicating that the smallest singular values might be lost.

If full SVD is needed, the following two condition numbers are useful for the analysis of the algorithm. They are provied for a developer/implementer who is familiar with the details of the method.

RWORK(4) = an estimate of the scaled condition number of the triangular factor in the first QR factorization. RWORK(5) = an estimate of the scaled condition number of the triangular factor in the second QR factorization. The following two parameters are computed if JOBT .EQ. 'T'. They are provided for a developer/implementer who is familiar with the details of the method. RWORK(6) = the entropy of A^* * A :: this is the Shannon entropy of diag(A^* * A) / Trace(A^* * A) taken as point in the probability simplex. RWORK(7) = the entropy of A * A^*. (See the description of RWORK(6).) If the call to ZGEJSV is a workspace query (indicated by LWORK=-1 or LRWORK=-1), then on exit RWORK(1) contains the required length of RWORK for the job parameters used in the call.

LRWORK Input parameter.

LRWORK is INTEGER

Length of RWORK to confirm proper allocation of workspace. LRWORK depends on the job:

1. If only the singular values are requested i.e. if LSAME(JOBU,'N') .AND. LSAME(JOBV,'N') then: 1.1. If LSAME(JOBT,'T') .OR. LSAME(JOBA,'F') .OR. LSAME(JOBA,'G'), then: LRWORK = max(7, 2 * M). 1.2. Otherwise, LRWORK = max(7, N). 2. If singular values with the right singular vectors are requested i.e. if (LSAME(JOBV,'V').OR.LSAME(JOBV,'J')) .AND. .NOT.(LSAME(JOBU,'U').OR.LSAME(JOBU,'F')) then: 2.1. If LSAME(JOBT,'T') .OR. LSAME(JOBA,'F') .OR. LSAME(JOBA,'G'), then LRWORK = max(7, 2 * M). 2.2. Otherwise, LRWORK = max(7, N). 3. If singular values with the left singular vectors are requested, i.e. if (LSAME(JOBU,'U').OR.LSAME(JOBU,'F')) .AND. .NOT.(LSAME(JOBV,'V').OR.LSAME(JOBV,'J')) then: 3.1. If LSAME(JOBT,'T') .OR. LSAME(JOBA,'F') .OR. LSAME(JOBA,'G'), then LRWORK = max(7, 2 * M). 3.2. Otherwise, LRWORK = max(7, N). 4. If singular values with both the left and the right singular vectors are requested, i.e. if (LSAME(JOBU,'U').OR.LSAME(JOBU,'F')) .AND. (LSAME(JOBV,'V').OR.LSAME(JOBV,'J')) then: 4.1. If LSAME(JOBT,'T') .OR. LSAME(JOBA,'F') .OR. LSAME(JOBA,'G'), then LRWORK = max(7, 2 * M). 4.2. Otherwise, LRWORK = max(7, N).

If, on entry, LRWORK = -1 or LWORK=-1, a workspace query is assumed and the length of RWORK is returned in RWORK(1).

IWORK Output parameter.

IWORK is INTEGER array, of dimension at least 4, that further depends

on the job:

1. If only the singular values are requested then: If (LSAME(JOBT,'T') .OR. LSAME(JOBA,'F') .OR. LSAME(JOBA,'G')) then the length of IWORK is N+M; otherwise the length of IWORK is N. 2. If the singular values and the right singular vectors are requested then: If (LSAME(JOBT,'T') .OR. LSAME(JOBA,'F') .OR. LSAME(JOBA,'G')) then the length of IWORK is N+M; otherwise the length of IWORK is N. 3. If the singular values and the left singular vectors are requested then: If (LSAME(JOBT,'T') .OR. LSAME(JOBA,'F') .OR. LSAME(JOBA,'G')) then the length of IWORK is N+M; otherwise the length of IWORK is N. 4. If the singular values with both the left and the right singular vectors are requested, then: 4.1. If LSAME(JOBV,'J') the length of IWORK is determined as follows: If (LSAME(JOBT,'T') .OR. LSAME(JOBA,'F') .OR. LSAME(JOBA,'G')) then the length of IWORK is N+M; otherwise the length of IWORK is N. 4.2. If LSAME(JOBV,'V') the length of IWORK is determined as follows: If (LSAME(JOBT,'T') .OR. LSAME(JOBA,'F') .OR. LSAME(JOBA,'G')) then the length of IWORK is 2*N+M; otherwise the length of IWORK is 2*N.

On exit, IWORK(1) = the numerical rank determined after the initial QR factorization with pivoting. See the descriptions of JOBA and JOBR. IWORK(2) = the number of the computed nonzero singular values IWORK(3) = if nonzero, a warning message: If IWORK(3).EQ.1 then some of the column norms of A were denormalized floats. The requested high accuracy is not warranted by the data. IWORK(4) = 1 or -1. If IWORK(4) .EQ. 1, then the procedure used A^* to do the job as specified by the JOB parameters. If the call to ZGEJSV is a workspace query (indicated by LWORK .EQ. -1 or LRWORK .EQ. -1), then on exit IWORK(1) contains the required length of IWORK for the job parameters used in the call.

INFO Output parameter.

INFO is INTEGER

< 0 : if INFO = -i, then the i-th argument had an illegal value. = 0 : successful exit; > 0 : ZGEJSV did not converge in the maximal allowed number of sweeps. The computed values may be inaccurate.

Related Information

For this routine in other precisions, please see *cgejsv*, *dgejsv* and *sgejsv*. It also exists with a native C interface as *LAPACKE_zgejsv*.

4.10.48 zgesdd

zgesdd computes the singular value decomposition (SVD) of a complex M-by-N matrix A, optionally computing the left and/or right singular vectors, by using divide-and-conquer method. The SVD is written

```
A = U * SIGMA * conjugate-transpose(V)
```

where SIGMA is an M-by-N matrix which is zero except for its min(m,n) diagonal elements, U is an M-by-M unitary matrix, and V is an N-by-N unitary matrix. The diagonal elements of SIGMA are the singular values of A; they are real and non-negative, and are returned in descending order. The first min(m,n) columns of U and V are the left and right singular vectors of A.

Note that the routine returns VT = V^H , not V.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgesdd(JOBZ, M, N, A, LDA, S, U, LDU, VT, LDVT, WORK, LWORK, RWORK,
                  IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgesdd_(const char *jobz, const armpl_int_t *m, const armpl_int_t *n,
             armpl_doublecomplex_t *a, const armpl_int_t *lda, double *s,
             armpl_doublecomplex_t *u, const armpl_int_t *ldu,
             armpl_doublecomplex_t *vt, const armpl_int_t *ldvt,
             armpl_doublecomplex_t *work, const armpl_int_t *lwork,
             double *rwork, armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

Specifies options for computing all or part of the matrix U: = 'A': all M columns of U and all N rows of V^H are returned in the arrays U and VT; = 'S': the first min(M, N) columns of U and the first min(M, N) rows of V^H are returned in the arrays U and VT; = 'O': If M >= N, the first N columns of U are overwritten in the array A and all rows of V^H are returned in the array VT; otherwise, all columns of U are returned in the array U and the first M rows of V^H are overwritten in the array A; = 'N': no columns of U or rows of V^H are computed.

M Input parameter.

M is INTEGER

The number of rows of the input matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the input matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, if JOBZ = 'O', A is overwritten with the first N columns of U (the left singular vectors, stored columnwise) if M >= N; A is overwritten with the first M rows of V^H (the right singular vectors, stored rowwise) otherwise. if JOBZ .ne. 'O', the contents of A are destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

S Output parameter.

S is DOUBLE PRECISION

S is an array, dimension (min(M, N)). The singular values of A, sorted so that S(i) >= S(i+1).

U Output parameter.

U is COMPLEX*16

U is an array, dimension (LDU,UCOL). UCOL = M if JOBZ = 'A' or JOBZ = 'O' and M < N; UCOL = min(M, N) if JOBZ = 'S'. If JOBZ = 'A' or JOBZ = 'O' and M < N, U contains the M-by-M unitary matrix U; if JOBZ = 'S', U contains the first min(M, N) columns of U (the left singular vectors, stored columnwise); if JOBZ = 'O' and M >= N, or JOBZ = 'N', U is not referenced.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U. LDU >= 1; if JOBZ = 'S' or 'A' or JOBZ = 'O' and M < N, LDU >= M.

VT Output parameter.

VT is COMPLEX*16

VT is an array, dimension (LDVT, N). If JOBZ = 'A' or JOBZ = 'O' and M >= N, VT contains the N-by-N unitary matrix V^H ; if JOBZ = 'S', VT contains the first min(M, N) rows of V^H (the right singular vectors, stored rowwise); if JOBZ = 'O' and M < N, or JOBZ = 'N', VT is not referenced.

LDVT Input parameter.

LDVT is INTEGER

The leading dimension of the array VT. LDVT >= 1; if JOBZ = 'A' or JOBZ = 'O' and M >= N, LDVT >= N; if JOBZ = 'S', LDVT >= min(M, N).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= 1. If LWORK = -1, a workspace query is assumed. The optimal size for the WORK array is calculated and stored in WORK(1), and no other work except argument checking is performed.

Let mx = max(M, N) and mn = min(M, N). If JOBZ = 'N', LWORK >= 2*mn + mx. If JOBZ = 'O', LWORK >= 2*mn*mn + 2*mn + mx. If JOBZ = 'S', LWORK >= mn*mn + 3*mn. If JOBZ = 'A', LWORK >= mn*mn + 2*mn + mx. These are not tight minimums in all cases; see comments inside code. For good performance, LWORK should generally be larger; a query is recommended.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (MAX(1,LRWORK)). Let mx = max(M, N) and mn = min(M, N). If JOBZ = 'N', LRWORK >= 5*mn (LAPACK <= 3.6 needs 7*mn); else if mx >> mn, LRWORK >= 5*mn*mn + 5*mn; else LRWORK >= max(5*mn*mn + 5*mn, 2*mx*mn + 2*mn*mn + mn).

IWORK Output parameter.

IWORK is INTEGER array, dimension (8*min(M, N))

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: The updating process of DBDSDC did not converge.

Related Information

For this routine in other precisions, please see *cgesdd*, *dgesdd* and *sgesdd*. It also exists with a native C interface as *LAPACKE_zgesdd*.

4.10.49 zgesvd

`zgesvd` computes the singular value decomposition (SVD) of a complex M-by-N matrix A, optionally computing the left and/or right singular vectors. The SVD is written

```
A = U * SIGMA * conjugate-transpose(V)
```

where SIGMA is an M-by-N matrix which is zero except for its min(m,n) diagonal elements, U is an M-by-M unitary matrix, and V is an N-by-N unitary matrix. The diagonal elements of SIGMA are the singular values of A; they are real and non-negative, and are returned in descending order. The first min(m,n) columns of U and V are the left and right singular vectors of A.

Note that the routine returns V^H , not V.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgesvd(JOBU, JOBVT, M, N, A, LDA, S, U, LDU, VT, LDVT, WORK, LWORK,
                  RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgesvd_(const char *jobu, const char *jobvt, const armpl_int_t *m,
             const armpl_int_t *n, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, double *s, armpl_doublecomplex_t *u,
             const armpl_int_t *ldu, armpl_doublecomplex_t *vt,
             const armpl_int_t *ldvt, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, double *rwork, armpl_int_t *info,
             ... );
```

Parameters

JOBU Input parameter.

`JOBU` is `CHARACTER*1`

Specifies options for computing all or part of the matrix U: = 'A': all `M` columns of `U` are returned in array U: = 'S': the first min(m,n) columns of `U` (the left singular vectors) are returned in the array U; = 'O': the first min(m,n) columns of `U` (the left singular vectors) are overwritten on the array A; = '`N`': no columns of `U` (no left singular vectors) are computed.

JOBVT Input parameter.

`JOBVT` is `CHARACTER*1`

Specifies options for computing all or part of the matrix V^H : = 'A': all `N` rows of V^H are returned in the array VT; = 'S': the first min(m,n) rows of V^H (the right singular vectors) are returned in the array VT; = 'O': the first min(m,n) rows of V^H (the right singular vectors) are overwritten on the array A; = '`N`': no rows of V^H (no right singular vectors) are computed.

`JOBVT` and `JOBU` cannot both be 'O'.

M Input parameter.

M is INTEGER

The number of rows of the input matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the input matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, if JOBU = 'O', A is overwritten with the first min(m,n) columns of U (the left singular vectors, stored columnwise); if JOBVT = 'O', A is overwritten with the first min(m,n) rows of V^H (the right singular vectors, stored rowwise); if JOBU .ne. 'O' and JOBVT .ne. 'O', the contents of A are destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

S Output parameter.

S is DOUBLE PRECISION

S is an array, dimension (min(M, N)). The singular values of A, sorted so that S(i) >= S(i+1).

U Output parameter.

U is COMPLEX*16

U is an array, dimension (LDU,UCOL). (LDU, M) if JOBU = 'A' or (LDU,min(M, N)) if JOBU = 'S'. If JOBU = 'A', U contains the M-by-M unitary matrix U; if JOBU = 'S', U contains the first min(m,n) columns of U (the left singular vectors, stored columnwise); if JOBU = 'N' or 'O', U is not referenced.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U. LDU >= 1; if JOBU = 'S' or 'A', LDU >= M.

VT Output parameter.

VT is COMPLEX*16

VT is an array, dimension (LDVT, N). If JOBVT = 'A', VT contains the N-by-N unitary matrix V^H ; if JOBVT = 'S', VT contains the first min(m,n) rows of V^H (the right singular vectors, stored rowwise); if JOBVT = 'N' or 'O', VT is not referenced.

LDVT Input parameter.

LDVT is INTEGER

The leading dimension of the array VT. LDVT >= 1; if JOBVT = 'A', LDVT >= N; if JOBVT = 'S', LDVT >= min(M, N).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= MAX(1,2*MIN(M, N)+MAX(M, N)). For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (5*min(M, N)). On exit, if INFO > 0, RWORK(1:MIN(M, N)-1) contains the unconverged superdiagonal elements of an upper bidiagonal matrix B whose diagonal is in S (not necessarily sorted). B satisfies A = U * B * VT, so it has the same singular values as A, and singular vectors related by U and VT.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if ZBDSQR did not converge, INFO specifies how many superdiagonals of an intermediate bidiagonal form B did not converge to zero. See the description of RWORK above for details.

Related Information

For this routine in other precisions, please see *cgesvd*, *dgesvd* and *sgesvd*. It also exists with a native C interface as *LAPACKE_zgesvd*.

4.10.50 zgesvdx

```
ZGESVDX computes the singular value decomposition (SVD) of a complex
 M-by-N matrix A, optionally computing the left and/or right singular
 vectors. The SVD is written

     A = U * SIGMA * transpose(V)

 where SIGMA is an M-by-N matrix which is zero except for its
 min(m,n) diagonal elements, U is an M-by-M unitary matrix, and
 V is an N-by-N unitary matrix.  The diagonal elements of SIGMA
 are the singular values of A; they are real and non-negative, and
 are returned in descending order.  The first min(m,n) columns of
 U and V are the left and right singular vectors of A.

 ZGESVDX uses an eigenvalue problem for obtaining the SVD, which
 allows for the computation of a subset of singular values and
 vectors. See DBDSVDX for details.

 Note that the routine returns V**T, not V.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zgesvdx(JOBU, JOBVT, RANGE, M, N, A, LDA, VL, VU, IL, IU, NS, S, U,
                   LDU, VT, LDVT, WORK, LWORK, RWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgesvdx_(const char *jobu, const char *jobvt, const char *range,
              const armpl_int_t *m, const armpl_int_t *n,
              armpl_doublecomplex_t *a, const armpl_int_t *lda,
              const double *vl, const double *vu, const armpl_int_t *il,
              const armpl_int_t *iu, armpl_int_t *ns, double *s,
              armpl_doublecomplex_t *u, const armpl_int_t *ldu,
              armpl_doublecomplex_t *vt, const armpl_int_t *ldvt,
              armpl_doublecomplex_t *work, const armpl_int_t *lwork,
              double *rwork, armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

JOBU Input parameter.

JOBU is CHARACTER*1

Specifies options for computing all or part of the matrix U: = 'V': the first min(m,n) columns of U (the left singular vectors) or as specified by RANGE are returned in the array U; = 'N': no columns of U (no left singular vectors) are computed.

JOBVT Input parameter.

JOBVT is CHARACTER*1

Specifies options for computing all or part of the matrix V^T : = 'V': the first min(m,n) rows of V^T (the right singular vectors) or as specified by RANGE are returned in the array VT; = 'N': no rows of V^T (no right singular vectors) are computed.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all singular values will be found. = 'V': all singular values in the half-open interval (VL,VU] will be found. = 'I': the IL-th through IU-th singular values will be found.

M Input parameter.

M is INTEGER

The number of rows of the input matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the input matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the contents of A are destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

VL Input parameter.

VL is DOUBLE PRECISION

If RANGE='V', the lower bound of the interval to be searched for singular values. VU > VL. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is DOUBLE PRECISION

If RANGE='V', the upper bound of the interval to be searched for singular values. VU > VL. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest singular value to be returned. 1 <= IL <= IU <= min(M, N), if min(M, N) > 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest singular value to be returned. 1 <= IL <= IU <= min(M, N), if min(M, N) > 0. Not referenced if RANGE = 'A' or 'V'.

NS Output parameter.

NS is INTEGER

The total number of singular values found, 0 <= NS <= min(M, N). If RANGE = 'A', NS = min(M, N); if RANGE = 'I', NS = IU-IL+1.

S Output parameter.

S is DOUBLE PRECISION

S is an array, dimension (min(M, N)). The singular values of A, sorted so that S(i) >= S(i+1).

U Output parameter.

U is COMPLEX*16

U is an array, dimension (LDU,UCOL). If JOBU = 'V', U contains columns of U (the left singular vectors, stored columnwise) as specified by RANGE; if JOBU = 'N', U is not referenced. Note: The user must ensure that UCOL >= NS; if RANGE = 'V', the exact value of NS is not known in advance and an upper bound must be used.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U. LDU >= 1; if JOBU = 'V', LDU >= M.

VT Output parameter.

VT is COMPLEX*16

VT is an array, dimension (LDVT, N). If JOBVT = 'V', VT contains the rows of V^T (the right singular vectors, stored rowwise) as specified by RANGE; if JOBVT = 'N', VT is not referenced. Note: The user must ensure that LDVT >= NS; if RANGE = 'V', the exact value of NS is not known in advance and an upper bound must be used.

LDVT Input parameter.

LDVT is INTEGER

The leading dimension of the array VT. LDVT >= 1; if JOBVT = 'V', LDVT >= NS (see above).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK;

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= MAX(1,MIN(M, N)*(MIN(M, N)+4)) for the paths (see comments inside the code): - PATH 1 (M much larger than N) - PATH 1t (N much larger than M) LWORK >= MAX(1,MIN(M, N)*2+MAX(M, N)) for the other paths. For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (MAX(1,LRWORK)). LRWORK >= MIN(M, N)*(MIN(M, N)*2+15*MIN(M, N)).

IWORK Output parameter.

IWORK is INTEGER array, dimension (12*MIN(M, N))

If INFO = 0, the first NS elements of IWORK are zero. If INFO > 0, then IWORK contains the indices of the eigenvectors that failed to converge in DBDSVDX/DSTEVX.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, then i eigenvectors failed to converge in DBDSVDX/DSTEVX. if INFO = N*2 + 1, an internal error occurred in DBDSVDX

Related Information

For this routine in other precisions, please see *cgesvdx*, *dgesvdx* and *sgesvdx*. It also exists with a native C interface as *LAPACKE_zgesvdx*.

4.10.51 zgesvj

zgesvj computes the singular value decomposition (SVD) of a complex M-by-N matrix A, where M >= N. The SVD of A is written as

```
                        [++]   [xx]   [x0]   [xx]
A = U * SIGMA * V^*,    [++] = [xx] * [ox] * [xx]
                        [++]   [xx]
```

where SIGMA is an N-by-N diagonal matrix, U is an M-by-N orthonormal matrix, and V is an N-by-N unitary matrix. The diagonal elements of SIGMA are the singular values of A. The columns of U and V are the left and the right singular vectors of A, respectively.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgesvj(JOBA, JOBU, JOBV, M, N, A, LDA, SVA, MV, V, LDV, CWORK,
                  LWORK, RWORK, LRWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgesvj_(const char *joba, const char *jobu, const char *jobv,
             const armpl_int_t *m, const armpl_int_t *n,
             armpl_doublecomplex_t *a, const armpl_int_t *lda, double *sva,
             const armpl_int_t *mv, armpl_doublecomplex_t *v,
             const armpl_int_t *ldv, armpl_doublecomplex_t *cwork,
             const armpl_int_t *lwork, double *rwork,
             const armpl_int_t *lrwork, armpl_int_t *info, ... );
```

Parameters

JOBA Input parameter.

JOBA is CHARACTER*1

Specifies the structure of A. = 'L': The input matrix A is lower triangular; = 'U': The input matrix A is upper triangular; = 'G': The input matrix A is general M-by-N matrix, M >= N.

JOBU Input parameter.

JOBU is CHARACTER*1

Specifies whether to compute the left singular vectors (columns of U): = 'U' or 'F': The left singular vectors corresponding to the nonzero singular values are computed and returned in the leading columns of A. See more details in the description of A. The default numerical orthogonality threshold is set to approximately TOL=CTOL*EPS, CTOL=SQRT(M), EPS=DLAMCH('E'). = 'C': Analogous to JOBU='U', except that user can control the level of numerical orthogonality of the computed left singular vectors. TOL can be set to TOL = CTOL*EPS, where CTOL is given on input in the array WORK. No CTOL smaller than ONE is allowed. CTOL greater than 1 / EPS is meaningless. The option 'C' can be used if M*EPS is satisfactory orthogonality of the computed left singular vectors, so CTOL=M could save few sweeps of Jacobi rotations. See the descriptions of A and WORK(1). = 'N': The matrix U is not computed. However, see the description of A.

JOBV Input parameter.

JOBV is CHARACTER*1

Specifies whether to compute the right singular vectors, that is, the matrix V: = 'V' or 'J': the matrix V is computed and returned in the array V = 'A' : the Jacobi rotations are applied to the MV-by-N array V. In other words, the right singular vector matrix V is not computed explicitly; instead it is applied to an MV-by-N matrix initially stored in the first MV rows of V. = 'N' : the matrix V is not computed and the array V is not referenced

M Input parameter.

M is INTEGER

The number of rows of the input matrix A. 1/DLAMCH('E') > M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the input matrix A. M >= N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, If JOBU .EQ. 'U' .OR. JOBU .EQ. 'C': If INFO .EQ. 0 : RANKA orthonormal columns of U are returned in the leading RANKA columns of the array A. Here RANKA <= N is the number of computed singular values of A that are above the underflow threshold DLAMCH('S'). The singular vectors corresponding to underflowed or zero singular values are not computed. The value of RANKA is returned in the array RWORK as RANKA=NINT(RWORK(2)). Also see the descriptions of SVA and RWORK. The computed columns of U are mutually numerically orthogonal up to

approximately TOL=SQRT(M)*EPS (default); or TOL=CTOL*EPS (JOBU.EQ.'C'), see the description of JOBU. If INFO .GT. 0, the procedure ZGESVJ did not converge in the given number of iterations (sweeps). In that case, the computed columns of U may not be orthogonal up to TOL. The output U (stored in A), SIGMA (given by the computed singular values in SVA(1:N)) and V is still a decomposition of the input matrix A in the sense that the residual || A - SCALE * U * SIGMA * V^* ||_2 / ||A||_2 is small. If JOBU .EQ. 'N': If INFO .EQ. 0 : Note that the left singular vectors are 'for free' in the one-sided Jacobi SVD algorithm. However, if only the singular values are needed, the level of numerical orthogonality of U is not an issue and iterations are stopped when the columns of the iterated matrix are numerically orthogonal up to approximately M*EPS. Thus, on exit, A contains the columns of U scaled with the corresponding singular values. If INFO .GT. 0 : the procedure ZGESVJ did not converge in the given number of iterations (sweeps).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

SVA Output parameter.

SVA is DOUBLE PRECISION

SVA is an array, dimension (N). On exit, If INFO .EQ. 0 : depending on the value SCALE = RWORK(1), we have: If SCALE .EQ. ONE: SVA(1:N) contains the computed singular values of A. During the computation SVA contains the Euclidean column norms of the iterated matrices in the array A. If SCALE .NE. ONE: The singular values of A are SCALE*SVA(1:N), and this factored representation is due to the fact that some of the singular values of A might underflow or overflow.

If INFO .GT. 0 : the procedure ZGESVJ did not converge in the given number of iterations (sweeps) and SCALE*SVA(1:N) may not be accurate.

MV Input parameter.

MV is INTEGER

If JOBV .EQ. 'A', then the product of Jacobi rotations in ZGESVJ is applied to the first MV rows of V. See the description of JOBV.

V Input and output parameter.

V is COMPLEX*16

V is an array, dimension (LDV, N). If JOBV = 'V', then V contains on exit the N-by-N matrix of the right singular vectors; If JOBV = 'A', then V contains the product of the computed right singular vector matrix and the initial matrix in the array V. If JOBV = 'N', then V is not referenced.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V, LDV .GE. 1. If JOBV .EQ. 'V', then LDV .GE. max(1, N). If JOBV .EQ. 'A', then LDV .GE. max(1, MV) .

CWORK Input and output parameter.

CWORK is COMPLEX*16

CWORK is an array, dimension (max(1, LWORK)). Used as workspace. If on entry LWORK .EQ. -1, then a workspace query is assumed and no computation is done; CWORK(1) is set to the minial (and optimal) length of CWORK.

LWORK Input parameter.

LWORK is INTEGER.

Length of CWORK, LWORK >= M+N.

RWORK Input and output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (max(6, LRWORK)). On entry, If JOBU .EQ. 'C' : RWORK(1) = CTOL, where CTOL defines the threshold for convergence. The process stops if all columns of A are mutually orthogonal up to CTOL*EPS, EPS=DLAMCH('E'). It is required that CTOL >= ONE, i.e. it is not allowed to force the routine to obtain orthogonality below EPSILON. On exit, RWORK(1) = SCALE is the scaling factor such that SCALE*SVA(1:N) are the computed singular values of A. (See description of SVA().) RWORK(2) = NINT(RWORK(2)) is the number of the computed nonzero singular values. RWORK(3) = NINT(RWORK(3)) is the number of the computed singular values that are larger than the underflow threshold. RWORK(4) = NINT(RWORK(4)) is the number of sweeps of Jacobi rotations needed for numerical convergence. RWORK(5) = max_{i.NE.j} |COS(A(:,i),A(:,j))| in the last sweep. This is useful information in cases when ZGESVJ did not converge, as it can be used to estimate whether the output is stil useful and for post festum analysis. RWORK(6) = the largest absolute value over all sines of the Jacobi rotation angles in the last sweep. It can be useful for a post festum analysis. If on entry LRWORK .EQ. -1, then a workspace query is assumed and no computation is done; RWORK(1) is set to the minial (and optimal) length of RWORK.

LRWORK Input parameter.

LRWORK is INTEGER

Length of RWORK, LRWORK >= MAX(6, N).

INFO Output parameter.

INFO is INTEGER

= 0 : successful exit. < 0 : if INFO = -i, then the i-th argument had an illegal value > 0 : ZGESVJ did not converge in the maximal allowed number (NSWEEP=30) of sweeps. The output may still be useful. See the description of RWORK.

Related Information

For this routine in other precisions, please see *cgesvj*, *dgesvj* and *sgesvj*. It also exists with a native C interface as *LAPACKE_zgesvj*.

4.10.52 zggsvd3

zggsvd3 computes the generalized singular value decomposition (GSVD) of an M-by-N complex matrix A and P-by-N complex matrix B:

```
U**H*A*Q = D1*( 0 R ),     V**H*B*Q = D2*( 0 R )
```

where U, V and Q are unitary matrices. Let K+L = the effective numerical rank of the matrix $(A^H, B^H)^H$, then R is a (K+L)-by-(K+L) nonsingular upper triangular matrix, D1 and D2 are M-by-(K+L) and P-by-(K+L) "diagonal" matrices and of the following structures, respectively:

If M-K-L >= 0,

```
                   K  L
     D1 =      K ( I  0 )
               L ( 0  C )
           M-K-L ( 0  0 )

                 K  L
     D2 =    L ( 0  S )
           P-L ( 0  0 )

                 N-K-L  K     L
( 0 R ) = K (    0    R11   R12 )
          L (    0     0    R22 )
```

where

```
 C = diag( ALPHA(K+1), ... , ALPHA(K+L) ),
S = diag( BETA(K+1),  ... , BETA(K+L) ),
C**2 + S**2 = I.

R is stored in A(1:K+L,N-K-L+1:N) on exit.
```

If M-K-L < 0,

```
                 K M-K K+L-M
     D1 =   K ( I  0    0   )
          M-K ( 0  C    0   )

                   K M-K K+L-M
     D2 =   M-K ( 0  S    0  )
          K+L-M ( 0  0    I  )
            P-L ( 0  0    0  )

                  N-K-L  K   M-K  K+L-M
( 0 R ) =     K ( 0    R11  R12  R13  )
            M-K ( 0     0   R22  R23  )
          K+L-M ( 0     0    0   R33  )
```

where

```
C = diag( ALPHA(K+1), ... , ALPHA(M) ),
S = diag( BETA(K+1),  ... , BETA(M) ),
C**2 + S**2 = I.

(R11 R12 R13 ) is stored in A(1:M, N-K-L+1:N), and R33 is stored
( 0  R22 R23 )
in B(M-K+1:L,N+M-K-L+1:N) on exit.
```

The routine computes C, S, R, and optionally the unitary transformation matrices U, V and Q.

In particular, if B is an N-by-N nonsingular matrix, then the GSVD of A and B implicitly gives the SVD of A*inv(B):

```
A*inv(B) = U*(D1*inv(D2))*V**H.
```

If $(A^H , B^H)^H$ has orthonormal columns, then the GSVD of A and B is also equal to the CS decomposition of A and B. Furthermore, the GSVD can be used to derive the solution of the eigenvalue problem:

```
A**H*A x = lambda* B**H*B x.
```

In some literature, the GSVD of A and B is presented in the form

```
U**H*A*X = ( 0 D1 ),   V**H*B*X = ( 0 D2 )
```

where U and V are orthogonal and X is nonsingular, and D1 and D2 are ‘’diagonal’‘. The former GSVD form can be converted to the latter form by taking the nonsingular matrix X as

```
X = Q*(  I   0    )
     (  0 inv(R) )
```

Syntax

Fortran specification:

```
use armpl_library
```

(continues on next page)

(continued from previous page)

```
subroutine zggsvd3(JOBU, JOBV, JOBQ, M, N, P, K, L, A, LDA, B, LDB, ALPHA,
                   BETA, U, LDU, V, LDV, Q, LDQ, WORK, LWORK, RWORK, IWORK,
                   INFO)
```

C specification:

```
#include "armpl.h"

void zggsvd3_(const char *jobu, const char *jobv, const char *jobq,
              const armpl_int_t *m, const armpl_int_t *n,
              const armpl_int_t *p, armpl_int_t *k, armpl_int_t *l,
              armpl_doublecomplex_t *a, const armpl_int_t *lda,
              armpl_doublecomplex_t *b, const armpl_int_t *ldb, double *alpha,
              double *beta, armpl_doublecomplex_t *u, const armpl_int_t *ldu,
              armpl_doublecomplex_t *v, const armpl_int_t *ldv,
              armpl_doublecomplex_t *q, const armpl_int_t *ldq,
              armpl_doublecomplex_t *work, const armpl_int_t *lwork,
              double *rwork, armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

JOBU Input parameter.

JOBU is CHARACTER*1

= 'U': Unitary matrix U is computed; = 'N': U is not computed.

JOBV Input parameter.

JOBV is CHARACTER*1

= 'V': Unitary matrix V is computed; = 'N': V is not computed.

JOBQ Input parameter.

JOBQ is CHARACTER*1

= 'Q': Unitary matrix Q is computed; = 'N': Q is not computed.

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrices A and B. N >= 0.

P Input parameter.

P is INTEGER

The number of rows of the matrix B. P >= 0.

K Output parameter.

K is INTEGER

L Output parameter.

L is INTEGER

On exit, K and L specify the dimension of the subblocks described in Purpose. K + L = effective numerical rank of $(A^H, B^H)^H$.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, A contains the triangular matrix R, or part of R. See Purpose for details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). On entry, the P-by-N matrix B. On exit, B contains part of the triangular matrix R if M-K-L < 0. See Purpose for details.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, P).

ALPHA Output parameter.

ALPHA is DOUBLE PRECISION

ALPHA is an array, dimension (N) .

BETA Output parameter.

BETA is DOUBLE PRECISION

BETA is an array, dimension (N). On exit, ALPHA and BETA contain the generalized singular value pairs of A and B; ALPHA(1:K) = 1, BETA(1:K) = 0, and if M-K-L >= 0, ALPHA(K+1:K+L) = C, BETA(K+1:K+L) = S, or if M-K-L < 0, ALPHA(K+1:M)=C, ALPHA(M+1:K+L)=0 BETA(K+1:M) =S, BETA(M+1:K+L) =1 and ALPHA(K+L+1:N) = 0 BETA(K+L+1:N) = 0

U Output parameter.

U is COMPLEX*16

U is an array, dimension (LDU, M). If JOBU = 'U', U contains the M-by-M unitary matrix U. If JOBU = 'N', U is not referenced.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U. LDU >= max(1, M) if JOBU = 'U'; LDU >= 1 otherwise.

V Output parameter.

V is COMPLEX*16

V is an array, dimension (LDV, P). If JOBV = 'V', V contains the P-by-P unitary matrix V. If JOBV = 'N', V is not referenced.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. LDV >= max(1, P) if JOBV = 'V'; LDV >= 1 otherwise.

Q Output parameter.

Q is COMPLEX*16

Q is an array, dimension (LDQ, N). If JOBQ = 'Q', Q contains the N-by-N unitary matrix Q. If JOBQ = 'N', Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, N) if JOBQ = 'Q'; LDQ >= 1 otherwise.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (2*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

On exit, IWORK stores the sorting information. More precisely, the following loop will sort ALPHA for I = K+1, min(M,K+L) swap ALPHA(I) and ALPHA(IWORK(I)) endfor such that ALPHA(1) >= ALPHA(2) >= ... >= ALPHA(N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = 1, the Jacobi-type procedure failed to converge. For further details, see subroutine ZTGSJA.

Related Information

For this routine in other precisions, please see *cggsvd3*, *dggsvd3* and *sggsvd3*. It also exists with a native C interface as *LAPACKE_zggsvd3*.

4.10.53 zggsvp3

zggsvp3 computes unitary matrices U, V and Q such that

```
                    N-K-L  K    L
U**H*A*Q =      K ( 0    A12  A13 )  if M-K-L >= 0;
                L ( 0     0   A23 )
            M-K-L ( 0     0    0  )

                 N-K-L  K    L
        =     K ( 0    A12  A13 )  if M-K-L < 0;
            M-K ( 0     0   A23 )

                 N-K-L  K    L
V**H*B*Q =   L ( 0     0   B13 )
           P-L ( 0     0    0  )
```

where the K-by-K matrix A12 and L-by-L matrix B13 are nonsingular upper triangular; A23 is L-by-L upper triangular if M-K-L >= 0, otherwise A23 is (M-K)-by-L upper trapezoidal. K+L = the effective numerical rank of the (M+P)-by-N matrix $(A^H, B^H)^H$.

This decomposition is the preprocessing step for computing the Generalized Singular Value Decomposition (GSVD), see subroutine ZGGSVD3.

Syntax

Fortran specification:

```
use armpl_library

subroutine zggsvp3(JOBU, JOBV, JOBQ, M, P, N, A, LDA, B, LDB, TOLA, TOLB, K,
                   L, U, LDU, V, LDV, Q, LDQ, IWORK, RWORK, TAU, WORK, LWORK,
                   INFO)
```

C specification:

```
#include "armpl.h"

void zggsvp3_(const char *jobu, const char *jobv, const char *jobq,
              const armpl_int_t *m, const armpl_int_t *p,
              const armpl_int_t *n, armpl_doublecomplex_t *a,
              const armpl_int_t *lda, armpl_doublecomplex_t *b,
              const armpl_int_t *ldb, const double *tola, const double *tolb,
              armpl_int_t *k, armpl_int_t *l, armpl_doublecomplex_t *u,
              const armpl_int_t *ldu, armpl_doublecomplex_t *v,
              const armpl_int_t *ldv, armpl_doublecomplex_t *q,
              const armpl_int_t *ldq, armpl_int_t *iwork, double *rwork,
              armpl_doublecomplex_t *tau, armpl_doublecomplex_t *work,
              const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

JOBU Input parameter.

JOBU is CHARACTER*1

= 'U': Unitary matrix U is computed; = 'N': U is not computed.

JOBV Input parameter.

JOBV is CHARACTER*1

= 'V': Unitary matrix V is computed; = 'N': V is not computed.

JOBQ Input parameter.

JOBQ is CHARACTER*1

= 'Q': Unitary matrix Q is computed; = 'N': Q is not computed.

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

P Input parameter.

P is INTEGER

The number of rows of the matrix B. P >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrices A and B. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, A contains the triangular (or trapezoidal) matrix described in the Purpose section.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). On entry, the P-by-N matrix B. On exit, B contains the triangular matrix described in the Purpose section.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, P).

TOLA Input parameter.

TOLA is DOUBLE PRECISION

TOLB Input parameter.

TOLB is DOUBLE PRECISION

TOLA and TOLB are the thresholds to determine the effective numerical rank of matrix B and a subblock of A. Generally, they are set to TOLA = MAX(M, N)*norm(A)*MAZHEPS, TOLB = MAX(P, N)*norm(B)*MAZHEPS. The size of TOLA and TOLB may affect the size of backward errors of the decomposition.

K Output parameter.

K is INTEGER

L Output parameter.

L is INTEGER

On exit, K and L specify the dimension of the subblocks described in Purpose section. K + L = effective numerical rank of $(A^H, B^H)^H$.

U Output parameter.

U is COMPLEX*16

U is an array, dimension (LDU, M). If JOBU = 'U', U contains the unitary matrix U. If JOBU = 'N', U is not referenced.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U. LDU >= max(1, M) if JOBU = 'U'; LDU >= 1 otherwise.

V Output parameter.

V is COMPLEX*16

V is an array, dimension (LDV, P). If JOBV = 'V', V contains the unitary matrix V. If JOBV = 'N', V is not referenced.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. LDV >= max(1, P) if JOBV = 'V'; LDV >= 1 otherwise.

Q Output parameter.

Q is COMPLEX*16

Q is an array, dimension (LDQ, N). If JOBQ = 'Q', Q contains the unitary matrix Q. If JOBQ = 'N', Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, N) if JOBQ = 'Q'; LDQ >= 1 otherwise.

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (2*N) .

TAU Output parameter.

TAU is COMPLEX*16

TAU is an array, dimension (N) .

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cggsvp3*, *dggsvp3* and *sggsvp3*. It also exists with a native C interface as *LAPACKE_zggsvp3*.

4.10.54 ztgsja

ztgsja computes the generalized singular value decomposition (GSVD) of two complex upper triangular (or trapezoidal) matrices A and B.

On entry, it is assumed that matrices A and B have the following forms, which may be obtained by the preprocessing subroutine ZGGSVP from a general M-by-N matrix A and P-by-N matrix B:

```
             N-K-L  K    L
A =    K ( 0    A12  A13 ) if M-K-L >= 0;
       L ( 0     0   A23 )
   M-K-L ( 0     0    0  )

            N-K-L  K    L
A =  K ( 0    A12  A13 ) if M-K-L < 0;
   M-K ( 0     0   A23 )

            N-K-L  K    L
B =  L ( 0     0   B13 )
   P-L ( 0     0    0  )
```

where the K-by-K matrix A12 and L-by-L matrix B13 are nonsingular upper triangular; A23 is L-by-L upper triangular if M-K-L >= 0, otherwise A23 is (M-K)-by-L upper trapezoidal.

On exit,

```
U**H *A*Q = D1*( 0 R ),    V**H *B*Q = D2*( 0 R ),
```

where U, V and Q are unitary matrices. R is a nonsingular upper triangular matrix, and D1 and D2 are ‘‘diagonal” matrices, which are of the following structures:

If M-K-L >= 0,

```
                  K  L
     D1 =     K ( I  0 )
              L ( 0  C )
          M-K-L ( 0  0 )

                 K  L
     D2 = L    ( 0  S )
          P-L  ( 0  0 )

               N-K-L  K    L
( 0 R ) = K (  0   R11  R12 ) K
          L (  0    0   R22 ) L
```

where

```
C = diag( ALPHA(K+1), ... , ALPHA(K+L) ),
S = diag( BETA(K+1),  ... , BETA(K+L) ),
C**2 + S**2 = I.

R is stored in A(1:K+L,N-K-L+1:N) on exit.
```

If M-K-L < 0,

```
            K M-K K+L-M
D1 =   K ( I  0    0   )
     M-K ( 0  C    0   )

              K M-K K+L-M
D2 =   M-K ( 0  S    0   )
     K+L-M ( 0  0    I   )
       P-L ( 0  0    0   )

             N-K-L  K   M-K  K+L-M
```

(0 R) = K (0 R11 R12 R13)

```
   M-K ( 0     0    R22  R23  )
K+L-M ( 0     0    0   R33  )
```

where C = diag(ALPHA(K+1), ... , ALPHA(M)), S = diag(BETA(K+1), ... , BETA(M)), C**2 + S**2 = I.

R = (R11 R12 R13) is stored in A(1:M, N-K-L+1:N) and R33 is stored

```
(  0  R22 R23 )
```

in B(M-K+1:L,N+M-K-L+1:N) on exit.

The computation of the unitary transformation matrices U, V or Q is optional. These matrices may either be formed explicitly, or they may be postmultiplied into input matrices U1, V1, or Q1.

Syntax

Fortran specification:

```
use armpl_library

subroutine ztgsja(JOBU, JOBV, JOBQ, M, P, N, K, L, A, LDA, B, LDB, TOLA, TOLB,
                  ALPHA, BETA, U, LDU, V, LDV, Q, LDQ, WORK, NCYCLE, INFO)
```

C specification:

```
#include "armpl.h"

void ztgsja_(const char *jobu, const char *jobv, const char *jobq,
             const armpl_int_t *m, const armpl_int_t *p, const armpl_int_t *n,
             const armpl_int_t *k, const armpl_int_t *l,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_doublecomplex_t *b, const armpl_int_t *ldb,
             const double *tola, const double *tolb, double *alpha,
             double *beta, armpl_doublecomplex_t *u, const armpl_int_t *ldu,
             armpl_doublecomplex_t *v, const armpl_int_t *ldv,
             armpl_doublecomplex_t *q, const armpl_int_t *ldq,
             armpl_doublecomplex_t *work, armpl_int_t *ncycle,
             armpl_int_t *info, ... );
```

Parameters

JOBU Input parameter.

JOBU is CHARACTER*1

= 'U': U must contain a unitary matrix U1 on entry, and the product U1* U is returned; = 'I': U is initialized to the unit matrix, and the unitary matrix U is returned; = 'N': U is not computed.

JOBV Input parameter.

JOBV is CHARACTER*1

= 'V': V must contain a unitary matrix V1 on entry, and the product V1* V is returned; = 'I': V is initialized to the unit matrix, and the unitary matrix V is returned; = 'N': V is not computed.

JOBQ Input parameter.

JOBQ is CHARACTER*1

= 'Q': Q must contain a unitary matrix Q1 on entry, and the product Q1* Q is returned; = 'I': Q is initialized to the unit matrix, and the unitary matrix Q is returned; = 'N': Q is not computed.

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

P Input parameter.

P is INTEGER

The number of rows of the matrix B. P >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrices A and B. N >= 0.

K Input parameter.

K is INTEGER

L Input parameter.

L is INTEGER

K and L specify the subblocks in the input matrices A and B: A23 = A(K+1:MIN(K+L, M),N-L+1:N) and B13 = B(1:L,,N-L+1:N) of A and B, whose GSVD is going to be computed by ZTGSJA. See Further Details.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, A(N-K+1:N,1:MIN(K+L, M)) contains the triangular matrix R or part of R. See Purpose for details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). On entry, the P-by-N matrix B. On exit, if necessary, B(M-K+1:L,N+M-K-L+1:N) contains a part of R. See Purpose for details.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, P).

TOLA Input parameter.

TOLA is DOUBLE PRECISION

TOLB Input parameter.

TOLB is DOUBLE PRECISION

TOLA and TOLB are the convergence criteria for the Jacobi- Kogbetliantz iteration procedure. Generally, they are the same as used in the preprocessing step, say TOLA = MAX(M, N)*norm(A)*MAZHEPS, TOLB = MAX(P, N)*norm(B)*MAZHEPS.

ALPHA Output parameter.

ALPHA is DOUBLE PRECISION

ALPHA is an array, dimension (N) .

BETA Output parameter.

BETA is DOUBLE PRECISION

BETA is an array, dimension (N). On exit, ALPHA and BETA contain the generalized singular value pairs of A and B; ALPHA(1:K) = 1, BETA(1:K) = 0, and if M-K-L >= 0, ALPHA(K+1:K+L) = diag(C), BETA(K+1:K+L) = diag(S), or if M-K-L < 0, ALPHA(K+1:M)= C, ALPHA(M+1:K+L)= 0 BETA(K+1:M) = S, BETA(M+1:K+L) = 1. Furthermore, if K+L < N, ALPHA(K+L+1:N) = 0 and BETA(K+L+1:N) = 0.

U Input and output parameter.

U is COMPLEX*16

U is an array, dimension (LDU, M). On entry, if JOBU = 'U', U must contain a matrix U1 (usually the unitary matrix returned by ZGGSVP). On exit, if JOBU = 'I', U contains the unitary matrix U; if JOBU = 'U', U contains the product U1*U. If JOBU = 'N', U is not referenced.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U. LDU >= max(1, M) if JOBU = 'U'; LDU >= 1 otherwise.

V Input and output parameter.

V is COMPLEX*16

V is an array, dimension (LDV, P). On entry, if JOBV = 'V', V must contain a matrix V1 (usually the unitary matrix returned by ZGGSVP). On exit, if JOBV = 'I', V contains the unitary matrix V; if JOBV = 'V', V contains the product V1*V. If JOBV = 'N', V is not referenced.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. LDV >= max(1, P) if JOBV = 'V'; LDV >= 1 otherwise.

Q Input and output parameter.

Q is COMPLEX*16

Q is an array, dimension (LDQ, N). On entry, if JOBQ = 'Q', Q must contain a matrix Q1 (usually the unitary matrix returned by ZGGSVP). On exit, if JOBQ = 'I', Q contains the unitary matrix Q; if JOBQ = 'Q', Q contains the product Q1*Q. If JOBQ = 'N', Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, N) if JOBQ = 'Q'; LDQ >= 1 otherwise.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

NCYCLE Output parameter.

NCYCLE is INTEGER

The number of cycles required for convergence.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. = 1: the procedure does not converge after MAXIT cycles.

Related Information

For this routine in other precisions, please see *ctgsja*, *dtgsja* and *stgsja*. It also exists with a native C interface as *LAPACKE_ztgsja*.

4.10.55 zungbr

zungbr generates one of the complex unitary matrices Q or P^H determined by ZGEBRD when reducing a complex matrix A to bidiagonal form: A = Q * B * P^H . Q and P^H are defined as products of elementary reflectors H(i) or G(i) respectively.

If VECT = 'Q', A is assumed to have been an M-by-K matrix, and Q is of order M: if m >= k, Q = H(1) H(2) . . . H(k) and zungbr returns the first n columns of Q, where m >= n >= k; if m < k, Q = H(1) H(2) . . . H(m-1) and zungbr returns Q as an M-by-M matrix.

If VECT = 'P', A is assumed to have been a K-by-N matrix, and P^H is of order N: if k < n, P^H = G(k) . . . G(2) G(1) and zungbr returns the first m rows of P^H , where n >= m >= k; if k >= n, P^H = G(n-1) . . . G(2) G(1) and zungbr returns P^H as an N-by-N matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine zungbr(VECT, M, N, K, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zungbr_(const char *vect, const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *k, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, const armpl_doublecomplex_t *tau,
             armpl_doublecomplex_t *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

VECT Input parameter.

VECT is CHARACTER*1

Specifies whether the matrix Q or the matrix P^H is required, as defined in the transformation applied by ZGEBRD: = 'Q': generate Q; = 'P': generate P^H .

M Input parameter.

M is INTEGER

The number of rows of the matrix Q or P^H to be returned. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix Q or P^H to be returned. N >= 0. If VECT = 'Q', M >= N >= min(M, K); if VECT = 'P', N >= M >= min(N, K).

K Input parameter.

K is INTEGER

If VECT = 'Q', the number of columns in the original M-by-K matrix reduced by ZGEBRD. If VECT = 'P', the number of rows in the original K-by-N matrix reduced by ZGEBRD. K >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the vectors which define the elementary reflectors, as returned by ZGEBRD. On exit, the M-by-N matrix Q or P^H .

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= M.

TAU Input parameter.

TAU is COMPLEX*16

TAU is an array, dimension. (min(M, K)) if VECT = 'Q' (min(N, K)) if VECT = 'P' TAU(i) must contain the scalar factor of the elementary reflector H(i) or G(i), which determines Q or P^H , as returned by ZGEBRD in its array argument TAUQ or TAUP.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,min(M, N)). For optimum performance LWORK >= min(M, N)*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cungbr*. It also exists with a native C interface as *LAPACKE_zungbr*.

4.10.56 zunmbr

If VECT = 'Q', zunmbr overwrites the general complex M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'C': Q^H * C C * Q^H

If VECT = 'P', zunmbr overwrites the general complex M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': P * C C * P TRANS = 'C': P^H * C C * P^H

Here Q and P^H are the unitary matrices determined by ZGEBRD when reducing a complex matrix A to bidiagonal form: A = Q * B * P^H . Q and P^H are defined as products of elementary reflectors H(i) and G(i) respectively.

Let nq = m if SIDE = 'L' and nq = n if SIDE = 'R'. Thus nq is the order of the unitary matrix Q or P^H that is applied.

If VECT = 'Q', A is assumed to have been an NQ-by-K matrix: if nq >= k, Q = H(1) H(2) . . . H(k); if nq < k, Q = H(1) H(2) . . . H(nq-1).

If VECT = 'P', A is assumed to have been a K-by-NQ matrix: if k < nq, P = G(1) G(2) . . . G(k); if k >= nq, P = G(1) G(2) . . . G(nq-1).

Syntax

Fortran specification:

```
use armpl_library

subroutine zunmbr(VECT, SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK,
                  LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zunmbr_(const char *vect, const char *side, const char *trans,
             const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *k,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_doublecomplex_t *tau, armpl_doublecomplex_t *c,
             const armpl_int_t *ldc, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

VECT Input parameter.

VECT is CHARACTER*1

= 'Q': apply Q or Q^H ; = 'P': apply P or P^H .

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q, Q^H , P or P^H from the Left; = 'R': apply Q, Q^H , P or P^H from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q or P; = 'C': Conjugate transpose, apply Q^H or P^H .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

If VECT = 'Q', the number of columns in the original matrix reduced by ZGEBRD. If VECT = 'P', the number of rows in the original matrix reduced by ZGEBRD. K >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension. (LDA,min(nq, K)) if VECT = 'Q' (LDA,nq) if VECT = 'P' The vectors which define the elementary reflectors H(i) and G(i), whose products determine the matrices Q and P, as returned by ZGEBRD.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If VECT = 'Q', LDA >= max(1,nq); if VECT = 'P', LDA >= max(1,min(nq, K)).

TAU Input parameter.

TAU is COMPLEX*16

TAU is an array, dimension (min(nq, K)). TAU(i) must contain the scalar factor of the elementary reflector H(i) or G(i) which determines Q or P, as returned by ZGEBRD in the array argument TAUQ or TAUP.

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^H * C or C*Q^H or C*Q or P* C or P^H * C or C*P or C*P^H .

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, N); if SIDE = 'R', LWORK >= max(1, M); if N = 0 or M = 0, LWORK >= 1. For optimum performance LWORK >= max(1,N*NB) if SIDE = 'L', and LWORK >= max(1,M*NB) if SIDE = 'R', where NB is the optimal blocksize. (NB = 0 if M = 0 or N = 0.)

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cunmbr*. It also exists with a native C interface as *LAPACKE_zunmbr*.

4.11 LAPACK symmetric eigenvalues routines

4.11.1 chbev

chbev computes all the eigenvalues and, optionally, eigenvectors of a complex Hermitian band matrix A.

Syntax

Fortran specification:

```
use armpl_library

subroutine chbev(JOBZ, UPLO, N, KD, AB, LDAB, W, Z, LDZ, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void chbev_(const char *jobz, const char *uplo, const armpl_int_t *n,
            const armpl_int_t *kd, armpl_singlecomplex_t *ab,
            const armpl_int_t *ldab, float *w, armpl_singlecomplex_t *z,
            const armpl_int_t *ldz, armpl_singlecomplex_t *work, float *rwork,
            armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the Hermitian band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, AB is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the first superdiagonal and the diagonal of the tridiagonal matrix T are returned in rows KD and KD+1 of AB, and if UPLO = 'L', the diagonal and first subdiagonal of T are returned in the first two rows of AB.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD + 1.

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the orthonormal eigenvectors of the matrix A, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (max(1,3*N-2)) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *zhbev*. It also exists with a native C interface as *LAPACKE_chbev*.

4.11.2 chbevd

chbevd computes all the eigenvalues and, optionally, eigenvectors of a complex Hermitian band matrix A. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine chbevd(JOBZ, UPLO, N, KD, AB, LDAB, W, Z, LDZ, WORK, LWORK, RWORK,
                  LRWORK, IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void chbevd_(const char *jobz, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *kd, armpl_singlecomplex_t *ab,
             const armpl_int_t *ldab, float *w, armpl_singlecomplex_t *z,
             const armpl_int_t *ldz, armpl_singlecomplex_t *work,
```

(continues on next page)

(continued from previous page)

```
                const armpl_int_t *lwork, float *rwork,
                const armpl_int_t *lrwork, armpl_int_t *iwork,
                const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the Hermitian band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, AB is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the first superdiagonal and the diagonal of the tridiagonal matrix T are returned in rows KD and KD+1 of AB, and if UPLO = 'L', the diagonal and first subdiagonal of T are returned in the first two rows of AB.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD + 1.

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the orthonormal eigenvectors of the matrix A, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If N <= 1, LWORK must be at least 1. If JOBZ = 'N' and N > 1, LWORK must be at least N. If JOBZ = 'V' and N > 1, LWORK must be at least 2*N**2.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array,. dimension (LRWORK) On exit, if INFO = 0, RWORK(1) returns the optimal LRWORK.

LRWORK Input parameter.

LRWORK is INTEGER

The dimension of array RWORK. If N <= 1, LRWORK must be at least 1. If JOBZ = 'N' and N > 1, LRWORK must be at least N. If JOBZ = 'V' and N > 1, LRWORK must be at least 1 + 5* N + 2*N**2.

If LRWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of array IWORK. If JOBZ = 'N' or N <= 1, LIWORK must be at least 1. If JOBZ = 'V' and N > 1, LIWORK must be at least 3 + 5* N .

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *zhbevd*. It also exists with a native C interface as *LAPACKE_chbevd*.

4.11.3 chbevx

chbevx computes selected eigenvalues and, optionally, eigenvectors of a complex Hermitian band matrix A. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine chbevx(JOBZ, RANGE, UPLO, N, KD, AB, LDAB, Q, LDQ, VL, VU, IL, IU,
                  ABSTOL, M, W, Z, LDZ, WORK, RWORK, IWORK, IFAIL, INFO)
```

C specification:

```
#include "armpl.h"

void chbevx_(const char *jobz, const char *range, const char *uplo,
             const armpl_int_t *n, const armpl_int_t *kd,
             armpl_singlecomplex_t *ab, const armpl_int_t *ldab,
             armpl_singlecomplex_t *q, const armpl_int_t *ldq,
             const float *vl, const float *vu, const armpl_int_t *il,
             const armpl_int_t *iu, const float *abstol, armpl_int_t *m,
             float *w, armpl_singlecomplex_t *z, const armpl_int_t *ldz,
             armpl_singlecomplex_t *work, float *rwork, armpl_int_t *iwork,
             armpl_int_t *ifail, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found; = 'V': all eigenvalues in the half-open interval (VL,VU] will be found; = 'I': the IL-th through IU-th eigenvalues will be found.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the Hermitian band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, AB is overwritten by values generated during the reduction to tridiagonal form.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD + 1.

Q Output parameter.

Q is COMPLEX

Q is an array, dimension (LDQ, N). If JOBZ = 'V', the N-by-N unitary matrix used in the reduction to tridiagonal form. If JOBZ = 'N', the array Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. If JOBZ = 'V', then LDQ >= max(1, N).

VL Input parameter.

VL is REAL

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is REAL

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is REAL

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing AB to tridiagonal form.

Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*SLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*SLAMCH('S').

See "Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy," by Demmel and Kahan, LAPACK Working Note #3.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is REAL

W is an array, dimension (N). The first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). If an eigenvector fails to converge, then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL. If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (7*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (5*N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (N)

If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, then i eigenvectors failed to converge. Their indices are stored in array IFAIL.

Related Information

For this routine in other precisions, please see *zhbevx*. It also exists with a native C interface as *LAPACKE_chbevx*.

4.11.4 chbtrd

chbtrd reduces a complex Hermitian band matrix A to real symmetric tridiagonal form T by a unitary similarity transformation: $Q^H * A * Q = T$.

Syntax

Fortran specification:

```
use armpl_library

subroutine chbtrd(VECT, UPLO, N, KD, AB, LDAB, D, E, Q, LDQ, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void chbtrd_(const char *vect, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *kd, armpl_singlecomplex_t *ab,
             const armpl_int_t *ldab, float *d, float *e,
             armpl_singlecomplex_t *q, const armpl_int_t *ldq,
             armpl_singlecomplex_t *work, armpl_int_t *info, ... );
```

Parameters

VECT Input parameter.

VECT is CHARACTER*1

= 'N': do not form Q; = 'V': form Q; = 'U': update a matrix X, by forming X*Q.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the Hermitian band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd). On exit, the diagonal elements of AB are overwritten by the diagonal elements of the tridiagonal matrix T; if KD > 0, the elements on the first superdiagonal (if UPLO = 'U') or the first subdiagonal (if UPLO = 'L') are overwritten by the off-diagonal elements of T; the rest of AB is overwritten by values generated during the reduction.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

D Output parameter.

D is REAL

D is an array, dimension (N). The diagonal elements of the tridiagonal matrix T.

E Output parameter.

E is REAL

E is an array, dimension (N-1). The off-diagonal elements of the tridiagonal matrix T: E(i) = T(i,i+1) if UPLO = 'U'; E(i) = T(i+1,i) if UPLO = 'L'.

Q Input and output parameter.

Q is COMPLEX

Q is an array, dimension (LDQ, N). On entry, if VECT = 'U', then Q must contain an N-by-N matrix X; if VECT = 'N' or 'V', then Q need not be set.

On exit: if VECT = 'V', Q contains the N-by-N unitary matrix Q; if VECT = 'U', Q contains the product X*Q; if VECT = 'N', the array Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= 1, and LDQ >= N if VECT = 'V' or 'U'.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zhbtrd*. It also exists with a native C interface as *LAPACKE_chbtrd*.

4.11.5 cheev

cheev computes all eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix A.

Syntax

Fortran specification:

```
use armpl_library

subroutine cheev(JOBZ, UPLO, N, A, LDA, W, WORK, LWORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cheev_(const char *jobz, const char *uplo, const armpl_int_t *n,
            armpl_singlecomplex_t *a, const armpl_int_t *lda, float *w,
            armpl_singlecomplex_t *work, const armpl_int_t *lwork,
            float *rwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A. On exit, if JOBZ = 'V', then if INFO = 0, A contains the orthonormal eigenvectors of the matrix A. If JOBZ = 'N', then on exit the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= max(1,2*N-1). For optimal efficiency, LWORK >= (NB+1)*N, where NB is the blocksize for CHETRD returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (max(1, 3*N-2)) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *zheev*. It also exists with a native C interface as *LAPACKE_cheev*.

4.11.6 cheevd

`cheevd` computes all eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix A. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine cheevd(JOBZ, UPLO, N, A, LDA, W, WORK, LWORK, RWORK, LRWORK, IWORK,
                  LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cheevd_(const char *jobz, const char *uplo, const armpl_int_t *n,
             armpl_singlecomplex_t *a, const armpl_int_t *lda, float *w,
             armpl_singlecomplex_t *work, const armpl_int_t *lwork,
             float *rwork, const armpl_int_t *lrwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A. On exit, if JOBZ = 'V', then if INFO = 0, A contains the orthonormal eigenvectors of the matrix A. If JOBZ = 'N', then on exit the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. If N <= 1, LWORK must be at least 1. If JOBZ = 'N' and N > 1, LWORK must be at least N + 1. If JOBZ = 'V' and N > 1, LWORK must be at least 2* N + N**2.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array,. dimension (LRWORK) On exit, if INFO = 0, RWORK(1) returns the optimal LRWORK.

LRWORK Input parameter.

LRWORK is INTEGER

The dimension of the array RWORK. If N <= 1, LRWORK must be at least 1. If JOBZ = 'N' and N > 1, LRWORK must be at least N. If JOBZ = 'V' and N > 1, LRWORK must be at least 1 + 5* N + 2*N**2.

If LRWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. If N <= 1, LIWORK must be at least 1. If JOBZ = 'N' and N > 1, LIWORK must be at least 1. If JOBZ = 'V' and N > 1, LIWORK must be at least 3 + 5*N.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i and JOBZ = 'N', then the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero; if INFO = i and JOBZ = 'V', then the algorithm failed to compute an eigenvalue while working on the submatrix lying in rows and columns INFO/(N+1) through mod(INFO,N+1).

Related Information

For this routine in other precisions, please see *zheevd*. It also exists with a native C interface as *LAPACKE_cheevd*.

4.11.7 cheevr

`cheevr` computes selected eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix A. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

`cheevr` first reduces the matrix A to tridiagonal form T with a call to CHETRD. Then, whenever possible, `cheevr` calls CSTEMR to compute the eigenspectrum using Relatively Robust Representations. CSTEMR computes eigenvalues by the dqds algorithm, while orthogonal eigenvectors are computed from various "good" L D L^T representations (also known as Relatively Robust Representations). Gram-Schmidt orthogonalization is avoided as far as possible. More specifically, the various steps of the algorithm are as follows.

For each unreduced block (submatrix) of T,

```
(a) Compute T - sigma I  = L D L^T, so that L and D
     define all the wanted eigenvalues to high relative accuracy.
     This means that small relative changes in the entries of D and L
     cause only small relative changes in the eigenvalues and
     eigenvectors. The standard (unfactored) representation of the
     tridiagonal matrix T does not have this property in general.
 (b) Compute the eigenvalues to suitable accuracy.
     If the eigenvectors are desired, the algorithm attains full
     accuracy of the computed eigenvalues only right before
     the corresponding vectors have to be computed, see steps c) and d).
 (c) For each cluster of close eigenvalues, select a new
     shift close to the cluster, find a new factorization, and refine
     the shifted eigenvalues to suitable accuracy.
 (d) For each eigenvalue with a large enough relative separation compute
     the corresponding eigenvector by forming a rank revealing twisted
     factorization. Go back to (c) for any clusters that remain.
```

The desired accuracy of the output can be specified by the input parameter ABSTOL.

For more details, see DSTEMR's documentation and: - Inderjit S. Dhillon and Beresford N. Parlett: "Multiple representations

```
 to compute orthogonal eigenvectors of symmetric tridiagonal matrices,"
Linear Algebra and its Applications, 387(1), pp. 1-28, August 2004.
```

- Inderjit Dhillon and Beresford Parlett: "Orthogonal Eigenvectors and

```
 Relative Gaps," SIAM Journal on Matrix Analysis and Applications, Vol. 25,
2004.  Also LAPACK Working Note 154.
```

- Inderjit Dhillon: "A new O(n^2) algorithm for the symmetric

```
 tridiagonal eigenvalue/eigenvector problem",
Computer Science Division Technical Report No. UCB/CSD-97-971,
UC Berkeley, May 1997.
```

Note 1 : `cheevr` calls CSTEMR when the full spectrum is requested on machines which conform to the ieee-754 floating point standard. `cheevr` calls SSTEBZ and CSTEIN on non-ieee machines and when partial spectrum requests are made.

Normal execution of CSTEMR may create NaNs and infinities and hence may abort due to a floating point exception in environments which do not handle NaNs and infinities in the ieee standard default manner.

Syntax

Fortran specification:

```
use armpl_library

subroutine cheevr(JOBZ, RANGE, UPLO, N, A, LDA, VL, VU, IL, IU, ABSTOL, M, W,
                  Z, LDZ, ISUPPZ, WORK, LWORK, RWORK, LRWORK, IWORK, LIWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void cheevr_(const char *jobz, const char *range, const char *uplo,
             const armpl_int_t *n, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, const float *vl, const float *vu,
             const armpl_int_t *il, const armpl_int_t *iu,
             const float *abstol, armpl_int_t *m, float *w,
             armpl_singlecomplex_t *z, const armpl_int_t *ldz,
             armpl_int_t *isuppz, armpl_singlecomplex_t *work,
             const armpl_int_t *lwork, float *rwork,
             const armpl_int_t *lrwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found. = 'V': all eigenvalues in the half-open interval (VL,VU] will be found. = 'I': the IL-th through IU-th eigenvalues will be found. For RANGE = 'V' or 'I' and IU - IL < N - 1, SSTEBZ and CSTEIN are called

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A. On exit, the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

VL Input parameter.

VL is REAL

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is REAL

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is REAL

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing A to tridiagonal form.

See "Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy," by Demmel and Kahan, LAPACK Working Note #3.

If high relative accuracy is important, set ABSTOL to SLAMCH('Safe minimum'). Doing so will guarantee that eigenvalues are computed to high relative accuracy when possible in future releases. The current code does not make any guarantees about high relative accuracy, but furutre releases will. See J. Barlow and J. Demmel, "Computing Accurate Eigensystems of Scaled Diagonally Dominant Matrices", LAPACK Working Note #7, for a discussion of which matrices define their eigenvalues to high relative accuracy.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is REAL

W is an array, dimension (N). The first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

ISUPPZ Output parameter.

ISUPPZ is INTEGER array, dimension (2*max(1, M))

The support of the eigenvectors in Z, i.e., the indices indicating the nonzero elements in Z. The i-th eigenvector is nonzero only in elements ISUPPZ(2*i-1) through ISUPPZ(2*i). This is an output of CSTEMR (tridiagonal matrix). The support of the eigenvectors of A is typically 1:N because of the unitary transformations applied by CUNMTR. Implemented only for RANGE = 'A' or 'I' and IU - IL = N - 1

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= max(1,2*N). For optimal efficiency, LWORK >= (NB+1)*N, where NB is the max of the blocksize for CHETRD and for CUNMTR as returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (MAX(1, LRWORK)). On exit, if INFO = 0, RWORK(1) returns the optimal (and minimal) LRWORK.

LRWORK Input parameter.

LRWORK is INTEGER

The length of the array RWORK. LRWORK >= max(1,24*N).

If LRWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal (and minimal) LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. LIWORK >= max(1,10*N).

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: Internal error

Related Information

For this routine in other precisions, please see *zheevr*. It also exists with a native C interface as *LAPACKE_cheevr*.

4.11.8 cheevx

`cheevx` computes selected eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix A. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine cheevx(JOBZ, RANGE, UPLO, N, A, LDA, VL, VU, IL, IU, ABSTOL, M, W,
                  Z, LDZ, WORK, LWORK, RWORK, IWORK, IFAIL, INFO)
```

C specification:

```
#include "armpl.h"

void cheevx_(const char *jobz, const char *range, const char *uplo,
             const armpl_int_t *n, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, const float *vl, const float *vu,
             const armpl_int_t *il, const armpl_int_t *iu,
             const float *abstol, armpl_int_t *m, float *w,
             armpl_singlecomplex_t *z, const armpl_int_t *ldz,
             armpl_singlecomplex_t *work, const armpl_int_t *lwork,
             float *rwork, armpl_int_t *iwork, armpl_int_t *ifail,
             armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found. = 'V': all eigenvalues in the half-open interval (VL,VU] will be found. = 'I': the IL-th through IU-th eigenvalues will be found.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A. On exit, the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

VL Input parameter.

VL is REAL

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is REAL

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is REAL

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing A to tridiagonal form.

Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*SLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*SLAMCH('S').

See "Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy," by Demmel and Kahan, LAPACK Working Note #3.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is REAL

W is an array, dimension (N). On normal exit, the first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). If an eigenvector fails to converge, then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL. If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= 1, when N <= 1; otherwise 2*N. For optimal efficiency, LWORK >= (NB+1)*N, where NB is the max of the blocksize for CHETRD and for CUNMTR as returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (7*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (5*N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (N)

If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, then i eigenvectors failed to converge. Their indices are stored in array IFAIL.

Related Information

For this routine in other precisions, please see *zheevx*. It also exists with a native C interface as *LAPACKE_cheevx*.

4.11.9 chetrd

chetrd reduces a complex Hermitian matrix A to real symmetric tridiagonal form T by a unitary similarity transformation: $Q^H * A * Q = T$.

Syntax

Fortran specification:

```
use armpl_library

subroutine chetrd(UPLO, N, A, LDA, D, E, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void chetrd_(const char *uplo, const armpl_int_t *n, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, float *d, float *e,
             armpl_singlecomplex_t *tau, armpl_singlecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, if UPLO = 'U', the diagonal and first superdiagonal of A are overwritten by the corresponding elements of the tridiagonal matrix T, and the elements above the first superdiagonal, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors; if UPLO = 'L', the diagonal and first subdiagonal of A are over- written by the corresponding elements of the tridiagonal matrix T, and the elements below the first subdiagonal, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

D Output parameter.

D is REAL

D is an array, dimension (N). The diagonal elements of the tridiagonal matrix T: D(i) = A(i,i).

E Output parameter.

E is REAL

E is an array, dimension (N-1). The off-diagonal elements of the tridiagonal matrix T: E(i) = A(i,i+1) if UPLO = 'U', E(i) = A(i+1,i) if UPLO = 'L'.

TAU Output parameter.

TAU is COMPLEX

TAU is an array, dimension (N-1). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= 1. For optimum performance LWORK >= N*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zhetrd*. It also exists with a native C interface as *LAPACKE_chetrd*.

4.11.10 chpev

chpev computes all the eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix in packed storage.

Syntax

Fortran specification:

```
use armpl_library

subroutine chpev(JOBZ, UPLO, N, AP, W, Z, LDZ, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void chpev_(const char *jobz, const char *uplo, const armpl_int_t *n,
            armpl_singlecomplex_t *ap, float *w, armpl_singlecomplex_t *z,
            const armpl_int_t *ldz, armpl_singlecomplex_t *work, float *rwork,
            armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the Hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n.

On exit, AP is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the diagonal and first superdiagonal of the tridiagonal matrix T overwrite the corresponding elements of A, and if UPLO = 'L', the diagonal and first subdiagonal of T overwrite the corresponding elements of A.

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the orthonormal eigenvectors of the matrix A, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (max(1, 2*N-1)) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (max(1, 3*N-2)) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *zhpev*. It also exists with a native C interface as *LAPACKE_chpev*.

4.11.11 chpevd

chpevd computes all the eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix A in packed storage. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like

the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine chpevd(JOBZ, UPLO, N, AP, W, Z, LDZ, WORK, LWORK, RWORK, LRWORK,
                  IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void chpevd_(const char *jobz, const char *uplo, const armpl_int_t *n,
             armpl_singlecomplex_t *ap, float *w, armpl_singlecomplex_t *z,
             const armpl_int_t *ldz, armpl_singlecomplex_t *work,
             const armpl_int_t *lwork, float *rwork,
             const armpl_int_t *lrwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the Hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n.

On exit, AP is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the diagonal and first superdiagonal of the tridiagonal matrix T overwrite the corresponding elements of A, and if UPLO = 'L', the diagonal and first subdiagonal of T overwrite the corresponding elements of A.

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the orthonormal eigenvectors of the matrix A, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the required LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of array WORK. If N <= 1, LWORK must be at least 1. If JOBZ = 'N' and N > 1, LWORK must be at least N. If JOBZ = 'V' and N > 1, LWORK must be at least 2*N.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the required sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (MAX(1, LRWORK)). On exit, if INFO = 0, RWORK(1) returns the required LRWORK.

LRWORK Input parameter.

LRWORK is INTEGER

The dimension of array RWORK. If N <= 1, LRWORK must be at least 1. If JOBZ = 'N' and N > 1, LRWORK must be at least N. If JOBZ = 'V' and N > 1, LRWORK must be at least 1 + 5* N + 2*N**2.

If LRWORK = -1, then a workspace query is assumed; the routine only calculates the required sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the required LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of array IWORK. If JOBZ = 'N' or N <= 1, LIWORK must be at least 1. If JOBZ = 'V' and N > 1, LIWORK must be at least 3 + 5*N.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the required sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *zhpevd*. It also exists with a native C interface as *LAPACKE_chpevd*.

4.11.12 chpevx

chpevx computes selected eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix A in packed storage. Eigenvalues/vectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine chpevx(JOBZ, RANGE, UPLO, N, AP, VL, VU, IL, IU, ABSTOL, M, W, Z,
                  LDZ, WORK, RWORK, IWORK, IFAIL, INFO)
```

C specification:

```
#include "armpl.h"

void chpevx_(const char *jobz, const char *range, const char *uplo,
             const armpl_int_t *n, armpl_singlecomplex_t *ap, const float *vl,
             const float *vu, const armpl_int_t *il, const armpl_int_t *iu,
             const float *abstol, armpl_int_t *m, float *w,
             armpl_singlecomplex_t *z, const armpl_int_t *ldz,
             armpl_singlecomplex_t *work, float *rwork, armpl_int_t *iwork,
             armpl_int_t *ifail, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found; = 'V': all eigenvalues in the half-open interval (VL,VU] will be found; = 'I': the IL-th through IU-th eigenvalues will be found.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the Hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n.

On exit, AP is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the diagonal and first superdiagonal of the tridiagonal matrix T overwrite the corresponding elements of A, and if UPLO = 'L', the diagonal and first subdiagonal of T overwrite the corresponding elements of A.

VL Input parameter.

VL is REAL

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is REAL

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is REAL

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing AP to tridiagonal form.

Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*SLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*SLAMCH('S').

See "Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy," by Demmel and Kahan, LAPACK Working Note #3.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the selected eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). If an eigenvector fails to converge, then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL. If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (7*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (5*N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (N)

If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, then i eigenvectors failed to converge. Their indices are stored in array IFAIL.

Related Information

For this routine in other precisions, please see *zhpevx*. It also exists with a native C interface as *LAPACKE_chpevx*.

4.11.13 chptrd

chptrd reduces a complex Hermitian matrix A stored in packed form to real symmetric tridiagonal form T by a unitary similarity transformation: $Q^H * A * Q = T$.

Syntax

Fortran specification:

```
use armpl_library

subroutine chptrd(UPLO, N, AP, D, E, TAU, INFO)
```

C specification:

```
#include "armpl.h"

void chptrd_(const char *uplo, const armpl_int_t *n,
             armpl_singlecomplex_t *ap, float *d, float *e,
             armpl_singlecomplex_t *tau, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the Hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n. On exit, if UPLO = 'U', the diagonal and first superdiagonal of A are overwritten by the corresponding elements of the tridiagonal matrix T, and the elements above the first superdiagonal, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors; if UPLO = 'L', the diagonal and first subdiagonal of A are over- written by the corresponding elements of the tridiagonal matrix T, and the elements below the first subdiagonal, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors. See Further Details.

D Output parameter.

D is REAL

D is an array, dimension (N). The diagonal elements of the tridiagonal matrix T: D(i) = A(i,i).

E Output parameter.

E is REAL

E is an array, dimension (N-1). The off-diagonal elements of the tridiagonal matrix T: E(i) = A(i,i+1) if UPLO = 'U', E(i) = A(i+1,i) if UPLO = 'L'.

TAU Output parameter.

TAU is COMPLEX

TAU is an array, dimension (N-1). The scalar factors of the elementary reflectors (see Further Details).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zhptrd*. It also exists with a native C interface as *LAPACKE_chptrd*.

4.11.14 cpteqr

`cpteqr` computes all eigenvalues and, optionally, eigenvectors of a symmetric positive definite tridiagonal matrix by first factoring the matrix using SPTTRF and then calling CBDSQR to compute the singular values of the bidiagonal factor.

This routine computes the eigenvalues of the positive definite tridiagonal matrix to high relative accuracy. This means that if the eigenvalues range over many orders of magnitude in size, then the small eigenvalues and corresponding eigenvectors will be computed more accurately than, for example, with the standard QR method.

The eigenvectors of a full or band positive definite Hermitian matrix can also be found if CHETRD, CHPTRD, or CHBTRD has been used to reduce this matrix to tridiagonal form. (The reduction to tridiagonal form, however, may preclude the possibility of obtaining high relative accuracy in the small eigenvalues of the original matrix, if these eigenvalues range over many orders of magnitude.)

Syntax

Fortran specification:

```
use armpl_library

subroutine cpteqr(COMPZ, N, D, E, Z, LDZ, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void cpteqr_(const char *compz, const armpl_int_t *n, float *d, float *e,
             armpl_singlecomplex_t *z, const armpl_int_t *ldz, float *work,
             armpl_int_t *info, ... );
```

Parameters

COMPZ Input parameter.

`COMPZ` is `CHARACTER*1`

= 'N': Compute eigenvalues only. = 'V': Compute eigenvectors of original Hermitian matrix also. Array `Z` contains the unitary matrix used to reduce the original matrix to tridiagonal form. = 'I': Compute eigenvectors of tridiagonal matrix also.

N Input parameter.

`N` is `INTEGER`

The order of the matrix. `N` >= 0.

D Input and output parameter.

`D` is `REAL`

`D` is an array, dimension (`N`). On entry, the n diagonal elements of the tridiagonal matrix. On normal exit, `D` contains the eigenvalues, in descending order.

E Input and output parameter.

`E` is `REAL`

`E` is an array, dimension (N-1). On entry, the (n-1) subdiagonal elements of the tridiagonal matrix. On exit, `E` has been destroyed.

Z Input and output parameter.

`Z` is `COMPLEX`

Z is an array, dimension (LDZ, N). On entry, if COMPZ = 'V', the unitary matrix used in the reduction to tridiagonal form. On exit, if COMPZ = 'V', the orthonormal eigenvectors of the original Hermitian matrix; if COMPZ = 'I', the orthonormal eigenvectors of the tridiagonal matrix. If INFO > 0 on exit, Z contains the eigenvectors associated with only the stored eigenvalues. If COMPZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if COMPZ = 'V' or 'I', LDZ >= max(1, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (4*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, and i is: <= N the Cholesky factorization of the matrix could not be performed because the i-th principal minor was not positive definite. > N the SVD algorithm failed to converge; if INFO = N+i, i off-diagonal elements of the bidiagonal factor did not converge to zero.

Related Information

For this routine in other precisions, please see *dpteqr*, *spteqr* and *zpteqr*. It also exists with a native C interface as *LAPACKE_cpteqr*.

4.11.15 cstedc

cstedc computes all eigenvalues and, optionally, eigenvectors of a symmetric tridiagonal matrix using the divide and conquer method. The eigenvectors of a full or band complex Hermitian matrix can also be found if CHETRD or CHPTRD or CHBTRD has been used to reduce this matrix to tridiagonal form.

This code makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none. See SLAED3 for details.

Syntax

Fortran specification:

```
use armpl_library

subroutine cstedc(COMPZ, N, D, E, Z, LDZ, WORK, LWORK, RWORK, LRWORK, IWORK,
                  LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cstedc_(const char *compz, const armpl_int_t *n, float *d, float *e,
             armpl_singlecomplex_t *z, const armpl_int_t *ldz,
             armpl_singlecomplex_t *work, const armpl_int_t *lwork,
             float *rwork, const armpl_int_t *lrwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

COMPZ Input parameter.

COMPZ is CHARACTER*1

= 'N': Compute eigenvalues only. = 'I': Compute eigenvectors of tridiagonal matrix also. = 'V': Compute eigenvectors of original Hermitian matrix also. On entry, Z contains the unitary matrix used to reduce the original matrix to tridiagonal form.

N Input parameter.

N is INTEGER

The dimension of the symmetric tridiagonal matrix. N >= 0.

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry, the diagonal elements of the tridiagonal matrix. On exit, if INFO = 0, the eigenvalues in ascending order.

E Input and output parameter.

E is REAL

E is an array, dimension (N-1). On entry, the subdiagonal elements of the tridiagonal matrix. On exit, E has been destroyed.

Z Input and output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, N). On entry, if COMPZ = 'V', then Z contains the unitary matrix used in the reduction to tridiagonal form. On exit, if INFO = 0, then if COMPZ = 'V', Z contains the orthonormal eigenvectors of the original Hermitian matrix, and if COMPZ = 'I', Z contains the orthonormal eigenvectors of the symmetric tridiagonal matrix. If COMPZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1. If eigenvectors are desired, then LDZ >= max(1, N).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If COMPZ = 'N' or 'I', or N <= 1, LWORK must be at least 1. If COMPZ = 'V' and N > 1, LWORK must be at least N*N. Note that for COMPZ = 'V', then if N is less than or equal to the minimum divide size, usually 25, then LWORK need only be 1.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (MAX(1, LRWORK)). On exit, if INFO = 0, RWORK(1) returns the optimal LRWORK.

LRWORK Input parameter.

LRWORK is INTEGER

The dimension of the array RWORK. If COMPZ = 'N' or N <= 1, LRWORK must be at least 1. If COMPZ = 'V' and N > 1, LRWORK must be at least 1 + 3* N + 2*N*lg N + 4*N**2 , where lg(N) = smallest integer k such that 2**k >= N. If COMPZ = 'I' and N > 1, LRWORK must be at least 1 + 4* N + 2*N**2 . Note that for COMPZ = 'I' or 'V', then if N is less than or equal to the minimum divide size, usually 25, then LRWORK need only be max(1,2*(N-1)).

If LRWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. If COMPZ = 'N' or N <= 1, LIWORK must be at least 1. If COMPZ = 'V' or N > 1, LIWORK must be at least 6 + 6* N + 5*N*lg N. If COMPZ = 'I' or N > 1, LIWORK must be at least 3 + 5* N . Note that for COMPZ = 'I' or 'V', then if N is less than or equal to the minimum divide size, usually 25, then LIWORK need only be 1.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: The algorithm failed to compute an eigenvalue while working on the submatrix lying in rows and columns INFO/(N+1) through mod(INFO,N+1).

Related Information

For this routine in other precisions, please see *dstedc*, *sstedc* and *zstedc*. It also exists with a native C interface as *LAPACKE_cstedc*.

4.11.16 cstegr

cstegr computes selected eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix T. Any such unreduced matrix has a well defined set of pairwise different real eigenvalues, the corresponding real eigenvectors are pairwise orthogonal.

The spectrum may be computed either completely or partially by specifying either an interval (VL,VU] or a range of indices IL:IU for the desired eigenvalues.

cstegr is a compatibility wrapper around the improved CSTEMR routine. See SSTEMR for further details.

One important change is that the ABSTOL parameter no longer provides any benefit and hence is no longer used.

Note : cstegr and CSTEMR work only on machines which follow IEEE-754 floating-point standard in their handling of infinities and NaNs. Normal execution may create these exceptiona values and hence may abort due to a floating point exception in environments which do not conform to the IEEE-754 standard.

Syntax

Fortran specification:

```
use armpl_library

subroutine cstegr(JOBZ, RANGE, N, D, E, VL, VU, IL, IU, ABSTOL, M, W, Z, LDZ,
                  ISUPPZ, WORK, LWORK, IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cstegr_(const char *jobz, const char *range, const armpl_int_t *n,
             float *d, float *e, const float *vl, const float *vu,
             const armpl_int_t *il, const armpl_int_t *iu,
             const float *abstol, armpl_int_t *m, float *w,
             armpl_singlecomplex_t *z, const armpl_int_t *ldz,
             armpl_int_t *isuppz, float *work, const armpl_int_t *lwork,
             armpl_int_t *iwork, const armpl_int_t *liwork, armpl_int_t *info,
             ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found. = 'V': all eigenvalues in the half-open interval (VL,VU] will be found. = 'I': the IL-th through IU-th eigenvalues will be found.

N Input parameter.

N is INTEGER

The order of the matrix. N >= 0.

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry, the N diagonal elements of the tridiagonal matrix T. On exit, D is overwritten.

E Input and output parameter.

E is REAL

E is an array, dimension (N). On entry, the (N-1) subdiagonal elements of the tridiagonal matrix T in elements 1 to N-1 of E. E(N) need not be set on input, but is used internally as workspace. On exit, E is overwritten.

VL Input parameter.

VL is REAL

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is REAL

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is REAL

Unused. Was the absolute error tolerance for the eigenvalues/eigenvectors in previous versions.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is REAL

W is an array, dimension (N). The first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', and if INFO = 0, then the first M columns of Z contain the orthonormal eigenvectors of the matrix T corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used. Supplying N columns is always safe.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', then LDZ >= max(1, N).

ISUPPZ Output parameter.

ISUPPZ is INTEGER array, dimension (2*max(1, M))

The support of the eigenvectors in Z, i.e., the indices indicating the nonzero elements in Z. The i-th computed eigenvector is nonzero only in elements ISUPPZ(2*i-1) through ISUPPZ(2*i). This is relevant in the case when the matrix is split. ISUPPZ is only accessed when JOBZ is 'V' and N > 0.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (LWORK). On exit, if INFO = 0, WORK(1) returns the optimal (and minimal) LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,18*N) if JOBZ = 'V', and LWORK >= max(1,12*N) if JOBZ = 'N'. If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size

of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (LIWORK)

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. LIWORK >= max(1,10*N) if the eigenvectors are desired, and LIWORK >= max(1,8*N) if only the eigenvalues are to be computed. If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the IWORK array, returns this value as the first entry of the IWORK array, and no error message related to LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

On exit, INFO = 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = 1X, internal error in SLARRE, if INFO = 2X, internal error in CLARRV. Here, the digit X = ABS(IINFO) < 10, where IINFO is the nonzero error code returned by SLARRE or CLARRV, respectively.

Related Information

For this routine in other precisions, please see *dstegr*, *sstegr* and *zstegr*. It also exists with a native C interface as *LAPACKE_cstegr*.

4.11.17 cstein

cstein computes the eigenvectors of a real symmetric tridiagonal matrix T corresponding to specified eigenvalues, using inverse iteration.

The maximum number of iterations allowed for each eigenvector is specified by an internal parameter MAXITS (currently set to 5).

Although the eigenvectors are real, they are stored in a complex array, which may be passed to CUNMTR or CUPMTR for back transformation to the eigenvectors of a complex Hermitian matrix which was reduced to tridiagonal form.

Syntax

Fortran specification:

```
use armpl_library

subroutine cstein(N, D, E, M, W, IBLOCK, ISPLIT, Z, LDZ, WORK, IWORK, IFAIL,
                  INFO)
```

C specification:

```
#include "armpl.h"

void cstein_(const armpl_int_t *n, const float *d, const float *e,
             const armpl_int_t *m, const float *w, const armpl_int_t *iblock,
             const armpl_int_t *isplit, armpl_singlecomplex_t *z,
             const armpl_int_t *ldz, float *work, armpl_int_t *iwork,
             armpl_int_t *ifail, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix. N >= 0.

D Input parameter.

D is REAL

D is an array, dimension (N). The n diagonal elements of the tridiagonal matrix T.

E Input parameter.

E is REAL

E is an array, dimension (N-1). The (n-1) subdiagonal elements of the tridiagonal matrix T, stored in elements 1 to N-1.

M Input parameter.

M is INTEGER

The number of eigenvectors to be found. 0 <= M <= N.

W Input parameter.

W is REAL

W is an array, dimension (N). The first M elements of W contain the eigenvalues for which eigenvectors are to be computed. The eigenvalues should be grouped by split-off block and ordered from smallest to largest within the block. (The output array W from SSTEBZ with ORDER = 'B' is expected here.)

IBLOCK Input parameter.

IBLOCK is INTEGER array, dimension (N)

The submatrix indices associated with the corresponding eigenvalues in W; IBLOCK(i)=1 if eigenvalue W(i) belongs to the first submatrix from the top, =2 if W(i) belongs to the second submatrix, etc. (The output array IBLOCK from SSTEBZ is expected here.)

ISPLIT Input parameter.

ISPLIT is INTEGER array, dimension (N)

The splitting points, at which T breaks up into submatrices. The first submatrix consists of rows/columns 1 to ISPLIT(1), the second of rows/columns ISPLIT(1)+1 through ISPLIT(2), etc. (The output array ISPLIT from SSTEBZ is expected here.)

Z Output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, M). The computed eigenvectors. The eigenvector associated with the eigenvalue W(i) is stored in the i-th column of Z. Any vector which fails to converge is set to its current iterate after MAXITS iterations. The imaginary parts of the eigenvectors are set to zero.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= max(1, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (5*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (M)

On normal exit, all elements of IFAIL are zero. If one or more eigenvectors fail to converge after MAXITS iterations, then their indices are stored in array IFAIL.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, then i eigenvectors failed to converge in MAXITS iterations. Their indices are stored in array IFAIL.

Related Information

For this routine in other precisions, please see *dstein*, *sstein* and *zstein*. It also exists with a native C interface as *LAPACKE_cstein*.

4.11.18 cstemr

cstemr computes selected eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix T. Any such unreduced matrix has a well defined set of pairwise different real eigenvalues, the corresponding real eigenvectors are pairwise orthogonal.

The spectrum may be computed either completely or partially by specifying either an interval (VL,VU] or a range of indices IL:IU for the desired eigenvalues.

Depending on the number of desired eigenvalues, these are computed either by bisection or the dqds algorithm. Numerically orthogonal eigenvectors are computed by the use of various suitable L D L^T factorizations near clusters of close eigenvalues (referred to as RRRs, Relatively Robust Representations). An informal sketch of the algorithm follows.

For each unreduced block (submatrix) of T,

```
(a) Compute T - sigma I  = L D L^T, so that L and D
     define all the wanted eigenvalues to high relative accuracy.
     This means that small relative changes in the entries of D and L
     cause only small relative changes in the eigenvalues and
     eigenvectors. The standard (unfactored) representation of the
     tridiagonal matrix T does not have this property in general.
 (b) Compute the eigenvalues to suitable accuracy.
     If the eigenvectors are desired, the algorithm attains full
     accuracy of the computed eigenvalues only right before
     the corresponding vectors have to be computed, see steps c) and d).
 (c) For each cluster of close eigenvalues, select a new
     shift close to the cluster, find a new factorization, and refine
     the shifted eigenvalues to suitable accuracy.
 (d) For each eigenvalue with a large enough relative separation compute
     the corresponding eigenvector by forming a rank revealing twisted
     factorization. Go back to (c) for any clusters that remain.
```

For more details, see: - Inderjit S. Dhillon and Beresford N. Parlett: “Multiple representations

```
 to compute orthogonal eigenvectors of symmetric tridiagonal matrices,"
Linear Algebra and its Applications, 387(1), pp. 1-28, August 2004.
```

- Inderjit Dhillon and Beresford Parlett: “Orthogonal Eigenvectors and

```
 Relative Gaps," SIAM Journal on Matrix Analysis and Applications, Vol. 25,
2004.  Also LAPACK Working Note 154.
```

- Inderjit Dhillon: “A new O(n^2) algorithm for the symmetric

```
 tridiagonal eigenvalue/eigenvector problem",
Computer Science Division Technical Report No. UCB/CSD-97-971,
UC Berkeley, May 1997.
```

Further Details 1.‘‘cstemr‘‘ works only on machines which follow IEEE-754 floating-point standard in their handling of infinities and NaNs. This permits the use of efficient inner loops avoiding a check for zero divisors.

2. LAPACK routines can be used to reduce a complex Hermitean matrix to real symmetric tridiagonal form.

(Any complex Hermitean tridiagonal matrix has real values on its diagonal and potentially complex numbers on its off-diagonals. By applying a similarity transform with an appropriate diagonal matrix diag(1,e^{i phy_1}, … , e^{i phy_{n-1}}), the complex Hermitean matrix can be transformed into a real symmetric matrix and complex arithmetic can be entirely avoided.)

While the eigenvectors of the real symmetric tridiagonal matrix are real, the eigenvectors of original complex Hermitean matrix have complex entries in general. Since LAPACK drivers overwrite the matrix data with the eigenvectors, `cstemr` accepts complex workspace to facilitate interoperability with CUNMTR or CUPMTR.

Syntax

Fortran specification:

```
use armpl_library

subroutine cstemr(JOBZ, RANGE, N, D, E, VL, VU, IL, IU, M, W, Z, LDZ, NZC,
                  ISUPPZ, TRYRAC, WORK, LWORK, IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cstemr_(const char *jobz, const char *range, const armpl_int_t *n,
             float *d, float *e, const float *vl, const float *vu,
             const armpl_int_t *il, const armpl_int_t *iu, armpl_int_t *m,
             float *w, armpl_singlecomplex_t *z, const armpl_int_t *ldz,
             const armpl_int_t *nzc, armpl_int_t *isuppz, armpl_int_t *tryrac,
             float *work, const armpl_int_t *lwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= ‘N’: Compute eigenvalues only; = ‘V’: Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= ‘A’: all eigenvalues will be found. = ‘V’: all eigenvalues in the half-open interval (VL,VU] will be found. = ‘I’: the IL-th through IU-th eigenvalues will be found.

N Input parameter.

N is INTEGER

The order of the matrix. N >= 0.

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry, the N diagonal elements of the tridiagonal matrix T. On exit, D is overwritten.

E Input and output parameter.

E is REAL

E is an array, dimension (N). On entry, the (N-1) subdiagonal elements of the tridiagonal matrix T in elements 1 to N-1 of E. E(N) need not be set on input, but is used internally as workspace. On exit, E is overwritten.

VL Input parameter.

VL is REAL

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is REAL

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0. Not referenced if RANGE = 'A' or 'V'.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is REAL

W is an array, dimension (N). The first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', and if INFO = 0, then the first M columns of Z contain the orthonormal eigenvectors of the matrix T corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and can be computed with a workspace query by setting NZC = -1, see below.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', then LDZ >= max(1, N).

NZC Input parameter.

NZC is INTEGER

The number of eigenvectors to be held in the array Z. If RANGE = 'A', then NZC >= max(1, N). If RANGE = 'V', then NZC >= the number of eigenvalues in (VL,VU]. If RANGE = 'I', then NZC >= IU-IL+1. If NZC =

-1, then a workspace query is assumed; the routine calculates the number of columns of the array Z that are needed to hold the eigenvectors. This value is returned as the first entry of the Z array, and no error message related to NZC is issued by XERBLA.

ISUPPZ Output parameter.

ISUPPZ is INTEGER array, dimension (2*max(1, M))

The support of the eigenvectors in Z, i.e., the indices indicating the nonzero elements in Z. The i-th computed eigenvector is nonzero only in elements ISUPPZ(2*i-1) through ISUPPZ(2*i). This is relevant in the case when the matrix is split. ISUPPZ is only accessed when JOBZ is 'V' and N > 0.

TRYRAC Input and output parameter.

TRYRAC is LOGICAL

If TRYRAC.EQ..TRUE., indicates that the code should check whether the tridiagonal matrix defines its eigenvalues to high relative accuracy. If so, the code uses relative-accuracy preserving algorithms that might be (a bit) slower depending on the matrix. If the matrix does not define its eigenvalues to high relative accuracy, the code can uses possibly faster algorithms. If TRYRAC.EQ..FALSE., the code is not required to guarantee relatively accurate eigenvalues and can use the fastest possible techniques. On exit, a .TRUE. TRYRAC will be set to .FALSE. if the matrix does not define its eigenvalues to high relative accuracy.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (LWORK). On exit, if INFO = 0, WORK(1) returns the optimal (and minimal) LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,18*N) if JOBZ = 'V', and LWORK >= max(1,12*N) if JOBZ = 'N'. If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (LIWORK)

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. LIWORK >= max(1,10*N) if the eigenvectors are desired, and LIWORK >= max(1,8*N) if only the eigenvalues are to be computed. If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the IWORK array, returns this value as the first entry of the IWORK array, and no error message related to LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

On exit, INFO = 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = 1X, internal error in SLARRE, if INFO = 2X, internal error in CLARRV. Here, the digit X = ABS(IINFO) < 10, where IINFO is the nonzero error code returned by SLARRE or CLARRV, respectively.

Related Information

For this routine in other precisions, please see *dstemr*, *sstemr* and *zstemr*. It also exists with a native C interface as *LAPACKE_cstemr*.

4.11.19 csteqr

csteqr computes all eigenvalues and, optionally, eigenvectors of a symmetric tridiagonal matrix using the implicit QL or QR method. The eigenvectors of a full or band complex Hermitian matrix can also be found if CHETRD or CHPTRD or CHBTRD has been used to reduce this matrix to tridiagonal form.

Syntax

Fortran specification:

```
use armpl_library

subroutine csteqr(COMPZ, N, D, E, Z, LDZ, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void csteqr_(const char *compz, const armpl_int_t *n, float *d, float *e,
             armpl_singlecomplex_t *z, const armpl_int_t *ldz, float *work,
             armpl_int_t *info, ... );
```

Parameters

COMPZ Input parameter.

COMPZ is CHARACTER*1

= 'N': Compute eigenvalues only. = 'V': Compute eigenvalues and eigenvectors of the original Hermitian matrix. On entry, Z must contain the unitary matrix used to reduce the original matrix to tridiagonal form. = 'I': Compute eigenvalues and eigenvectors of the tridiagonal matrix. Z is initialized to the identity matrix.

N Input parameter.

N is INTEGER

The order of the matrix. N >= 0.

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry, the diagonal elements of the tridiagonal matrix. On exit, if INFO = 0, the eigenvalues in ascending order.

E Input and output parameter.

E is REAL

E is an array, dimension (N-1). On entry, the (n-1) subdiagonal elements of the tridiagonal matrix. On exit, E has been destroyed.

Z Input and output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, N). On entry, if COMPZ = 'V', then Z contains the unitary matrix used in the reduction to tridiagonal form. On exit, if INFO = 0, then if COMPZ = 'V', Z contains the orthonormal eigenvectors of the original Hermitian matrix, and if COMPZ = 'I', Z contains the orthonormal eigenvectors of the symmetric tridiagonal matrix. If COMPZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if eigenvectors are desired, then LDZ >= max(1, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (max(1,2*N-2)). If COMPZ = 'N', then WORK is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: the algorithm has failed to find all the eigenvalues in a total of 30* N iterations; if INFO = i, then i elements of E have not converged to zero; on exit, D and E contain the elements of a symmetric tridiagonal matrix which is unitarily similar to the original matrix.

Related Information

For this routine in other precisions, please see *dsteqr*, *ssteqr* and *zsteqr*. It also exists with a native C interface as *LAPACKE_csteqr*.

4.11.20 cungtr

cungtr generates a complex unitary matrix Q which is defined as the product of n-1 elementary reflectors of order N, as returned by CHETRD:

if UPLO = 'U', Q = H(n-1) . . . H(2) H(1),

if UPLO = 'L', Q = H(1) H(2) . . . H(n-1).

Syntax

Fortran specification:

```
use armpl_library

subroutine cungtr(UPLO, N, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cungtr_(const char *uplo, const armpl_int_t *n, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, const armpl_singlecomplex_t *tau,
             armpl_singlecomplex_t *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A contains elementary reflectors from CHETRD; = 'L': Lower triangle of A contains elementary reflectors from CHETRD.

N Input parameter.

N is INTEGER

The order of the matrix Q. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the vectors which define the elementary reflectors, as returned by CHETRD. On exit, the N-by-N unitary matrix Q.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= N.

TAU Input parameter.

TAU is COMPLEX

TAU is an array, dimension (N-1). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by CHETRD.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= N-1. For optimum performance LWORK >= (N-1)*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zungtr*. It also exists with a native C interface as *LAPACKE_cungtr*.

4.11.21 cunm22

```
CUNM22 overwrites the general complex M-by-N matrix C with

                 SIDE = 'L'     SIDE = 'R'
 TRANS = 'N':      Q * C          C * Q
 TRANS = 'C':      Q**H * C       C * Q**H

 where Q is a complex unitary matrix of order NQ, with NQ = M if
 SIDE = 'L' and NQ = N if SIDE = 'R'.
 The unitary matrix Q processes a 2-by-2 block structure

        [  Q11  Q12  ]
    Q = [            ]
        [  Q21  Q22  ],

 where Q12 is an N1-by-N1 lower triangular matrix and Q21 is an
 N2-by-N2 upper triangular matrix.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine cunm22(SIDE, TRANS, M, N, N1, N2, Q, LDQ, C, LDC, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void cunm22_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *n1,
             const armpl_int_t *n2, const armpl_singlecomplex_t *q,
             const armpl_int_t *ldq, armpl_singlecomplex_t *c,
             const armpl_int_t *ldc, armpl_singlecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left; = 'R': apply Q or Q^H from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply Q (No transpose); = 'C': apply Q^H (Conjugate transpose).

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

N1 Input parameter.

N2 Input parameter.

N1 is INTEGER

N2 is INTEGER The dimension of Q12 and Q21, respectively. N1, N2 >= 0. The following requirement must be satisfied: N1 + N2 = M if SIDE = 'L' and N1 + N2 = N if SIDE = 'R'.

Q Input parameter.

Q is COMPLEX

Q is an array, dimension. (LDQ, M) if SIDE = 'L' (LDQ, N) if SIDE = 'R'

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, M) if SIDE = 'L'; LDQ >= max(1, N) if SIDE = 'R'.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^H * C or C*Q^H or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, N); if SIDE = 'R', LWORK >= max(1, M). For optimum performance LWORK >= M*N.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zunm22*.

4.11.22 cunmtr

cunmtr overwrites the general complex M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'C': Q^H * C C * Q^H

where Q is a complex unitary matrix of order nq, with nq = m if SIDE = 'L' and nq = n if SIDE = 'R'. Q is defined as the product of nq-1 elementary reflectors, as returned by CHETRD:

if UPLO = 'U', Q = H(nq-1) . . . H(2) H(1);

if UPLO = 'L', Q = H(1) H(2) . . . H(nq-1).

Syntax

Fortran specification:

```
use armpl_library

subroutine cunmtr(SIDE, UPLO, TRANS, M, N, A, LDA, TAU, C, LDC, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void cunmtr_(const char *side, const char *uplo, const char *trans,
             const armpl_int_t *m, const armpl_int_t *n,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const armpl_singlecomplex_t *tau, armpl_singlecomplex_t *c,
             const armpl_int_t *ldc, armpl_singlecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left; = 'R': apply Q or Q^H from the Right.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A contains elementary reflectors from CHETRD; = 'L': Lower triangle of A contains elementary reflectors from CHETRD.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'C': Conjugate transpose, apply Q^H .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension. (LDA, M) if SIDE = 'L' (LDA, N) if SIDE = 'R' The vectors which define the elementary reflectors, as returned by CHETRD.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M) if SIDE = 'L'; LDA >= max(1, N) if SIDE = 'R'.

TAU Input parameter.

TAU is COMPLEX

TAU is an array, dimension. (M-1) if SIDE = 'L' (N-1) if SIDE = 'R' TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by CHETRD.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^H * C or C*Q^H or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, N); if SIDE = 'R', LWORK >= max(1, M). For optimum performance LWORK >= N*NB if SIDE = 'L', and LWORK >=M*NB if SIDE = 'R', where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zunmtr*. It also exists with a native C interface as *LAPACKE_cunmtr*.

4.11.23 cupgtr

cupgtr generates a complex unitary matrix Q which is defined as the product of n-1 elementary reflectors H(i) of order n, as returned by CHPTRD using packed storage:

if UPLO = 'U', Q = H(n-1) . . . H(2) H(1),

if UPLO = 'L', Q = H(1) H(2) . . . H(n-1).

Syntax

Fortran specification:

```
use armpl_library

subroutine cupgtr(UPLO, N, AP, TAU, Q, LDQ, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void cupgtr_(const char *uplo, const armpl_int_t *n,
             const armpl_singlecomplex_t *ap,
             const armpl_singlecomplex_t *tau, armpl_singlecomplex_t *q,
             const armpl_int_t *ldq, armpl_singlecomplex_t *work,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangular packed storage used in previous call to CHPTRD; = 'L': Lower triangular packed storage used in previous call to CHPTRD.

N Input parameter.

N is INTEGER

The order of the matrix Q. N >= 0.

AP Input parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). The vectors which define the elementary reflectors, as returned by CHPTRD.

TAU Input parameter.

TAU is COMPLEX

TAU is an array, dimension (N-1). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by CHPTRD.

Q Output parameter.

Q is COMPLEX

Q is an array, dimension (LDQ, N). The N-by-N unitary matrix Q.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, N).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (N-1) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zupgtr*. It also exists with a native C interface as *LAPACKE_cupgtr*.

4.11.24 cupmtr

cupmtr overwrites the general complex M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'C': Q^H * C C * Q^H

where Q is a complex unitary matrix of order nq, with nq = m if SIDE = 'L' and nq = n if SIDE = 'R'. Q is defined as the product of nq-1 elementary reflectors, as returned by CHPTRD using packed storage:

if UPLO = 'U', Q = H(nq-1) . . . H(2) H(1);

if UPLO = 'L', Q = H(1) H(2) . . . H(nq-1).

Syntax

Fortran specification:

```
use armpl_library

subroutine cupmtr(SIDE, UPLO, TRANS, M, N, AP, TAU, C, LDC, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void cupmtr_(const char *side, const char *uplo, const char *trans,
             const armpl_int_t *m, const armpl_int_t *n,
             const armpl_singlecomplex_t *ap,
             const armpl_singlecomplex_t *tau, armpl_singlecomplex_t *c,
             const armpl_int_t *ldc, armpl_singlecomplex_t *work,
             armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left; = 'R': apply Q or Q^H from the Right.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangular packed storage used in previous call to CHPTRD; = 'L': Lower triangular packed storage used in previous call to CHPTRD.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'C': Conjugate transpose, apply Q^H .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

AP Input parameter.

AP is COMPLEX

AP is an array, dimension. (M*(M+1)/2) if SIDE = 'L' (N*(N+1)/2) if SIDE = 'R' The vectors which define the elementary reflectors, as returned by CHPTRD. AP is modified by the routine but restored on exit.

TAU Input parameter.

TAU is COMPLEX

TAU is an array, dimension (M-1) if SIDE = 'L'. or (N-1) if SIDE = 'R' TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by CHPTRD.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^H * C or C*Q^H or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension. (N) if SIDE = 'L' (M) if SIDE = 'R'

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zupmtr*. It also exists with a native C interface as *LAPACKE_cupmtr*.

4.11.25 ddisna

ddisna computes the reciprocal condition numbers for the eigenvectors of a real symmetric or complex Hermitian matrix or for the left or right singular vectors of a general m-by-n matrix. The reciprocal condition number is the 'gap' between the corresponding eigenvalue or singular value and the nearest other one.

The bound on the error, measured by angle in radians, in the I-th computed vector is given by

```
DLAMCH( 'E' ) * ( ANORM / SEP( I ) )
```

where ANORM = 2-norm(A) = max(abs(D(j))). SEP(I) is not allowed to be smaller than DLAMCH('E')*ANORM in order to limit the size of the error bound.

ddisna may also be used to compute error bounds for eigenvectors of the generalized symmetric definite eigenproblem.

Syntax

Fortran specification:

```
use armpl_library

subroutine ddisna(JOB, M, N, D, SEP, INFO)
```

C specification:

```
#include "armpl.h"

void ddisna_(const char *job, const armpl_int_t *m, const armpl_int_t *n,
             const double *d, double *sep, armpl_int_t *info, ... );
```

Parameters

JOB Input parameter.

JOB is CHARACTER*1

Specifies for which problem the reciprocal condition numbers should be computed: = 'E': the eigenvectors of a symmetric/Hermitian matrix; = 'L': the left singular vectors of a general matrix; = 'R': the right singular vectors of a general matrix.

M Input parameter.

M is INTEGER

The number of rows of the matrix. M >= 0.

N Input parameter.

N is INTEGER

If JOB = 'L' or 'R', the number of columns of the matrix, in which case N >= 0. Ignored if JOB = 'E'.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (M) if JOB = 'E'. dimension (min(M, N)) if JOB = 'L' or 'R' The eigenvalues (if JOB = 'E') or singular values (if JOB = 'L' or 'R') of the matrix, in either increasing or decreasing order. If singular values, they must be non-negative.

SEP Output parameter.

SEP is DOUBLE PRECISION

SEP is an array, dimension (M) if JOB = 'E'. dimension (min(M, N)) if JOB = 'L' or 'R' The reciprocal condition numbers of the vectors.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *sdisna*. It also exists with a native C interface as *LAPACKE_ddisna*.

4.11.26 dopgtr

dopgtr generates a real orthogonal matrix Q which is defined as the product of n-1 elementary reflectors H(i) of order n, as returned by DSPTRD using packed storage:

if UPLO = 'U', Q = H(n-1) . . . H(2) H(1),

if UPLO = 'L', Q = H(1) H(2) . . . H(n-1).

Syntax

Fortran specification:

```
use armpl_library

subroutine dopgtr(UPLO, N, AP, TAU, Q, LDQ, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dopgtr_(const char *uplo, const armpl_int_t *n, const double *ap,
             const double *tau, double *q, const armpl_int_t *ldq,
             double *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangular packed storage used in previous call to DSPTRD; = 'L': Lower triangular packed storage used in previous call to DSPTRD.

N Input parameter.

N is INTEGER

The order of the matrix Q. N >= 0.

AP Input parameter.

AP is DOUBLE PRECISION

AP is an array, dimension (N*(N+1)/2). The vectors which define the elementary reflectors, as returned by DSPTRD.

TAU Input parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (N-1). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by DSPTRD.

Q Output parameter.

Q is DOUBLE PRECISION

Q is an array, dimension (LDQ, N). The N-by-N orthogonal matrix Q.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (N-1) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *sopgtr*. It also exists with a native C interface as *LAPACKE_dopgtr*.

4.11.27 dopmtr

dopmtr overwrites the general real M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'T': Q^T * C C * Q^T

where Q is a real orthogonal matrix of order nq, with nq = m if SIDE = 'L' and nq = n if SIDE = 'R'. Q is defined as the product of nq-1 elementary reflectors, as returned by DSPTRD using packed storage:

if UPLO = 'U', Q = H(nq-1) . . . H(2) H(1);

if UPLO = 'L', Q = H(1) H(2) . . . H(nq-1).

Syntax

Fortran specification:

```
use armpl_library

subroutine dopmtr(SIDE, UPLO, TRANS, M, N, AP, TAU, C, LDC, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dopmtr_(const char *side, const char *uplo, const char *trans,
             const armpl_int_t *m, const armpl_int_t *n, const double *ap,
             const double *tau, double *c, const armpl_int_t *ldc,
             double *work, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangular packed storage used in previous call to DSPTRD; = 'L': Lower triangular packed storage used in previous call to DSPTRD.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'T': Transpose, apply Q^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

AP Input parameter.

AP is DOUBLE PRECISION

AP is an array, dimension. (M*(M+1)/2) if SIDE = 'L' (N*(N+1)/2) if SIDE = 'R' The vectors which define the elementary reflectors, as returned by DSPTRD. AP is modified by the routine but restored on exit.

TAU Input parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (M-1) if SIDE = 'L'. or (N-1) if SIDE = 'R' TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by DSPTRD.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension. (N) if SIDE = 'L' (M) if SIDE = 'R'

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *sopmtr*. It also exists with a native C interface as *LAPACKE_dopmtr*.

4.11.28 dorgtr

dorgtr generates a real orthogonal matrix Q which is defined as the product of n-1 elementary reflectors of order N, as returned by DSYTRD:

if UPLO = 'U', Q = H(n-1) . . . H(2) H(1),

if UPLO = 'L', Q = H(1) H(2) . . . H(n-1).

Syntax

Fortran specification:

```
use armpl_library

subroutine dorgtr(UPLO, N, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dorgtr_(const char *uplo, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, const double *tau, double *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A contains elementary reflectors from DSYTRD; = 'L': Lower triangle of A contains elementary reflectors from DSYTRD.

N Input parameter.

N is INTEGER

The order of the matrix Q. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the vectors which define the elementary reflectors, as returned by DSYTRD. On exit, the N-by-N orthogonal matrix Q.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TAU Input parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (N-1). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by DSYTRD.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,N-1). For optimum performance LWORK >= (N-1)*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *sorgtr*. It also exists with a native C interface as *LAPACKE_dorgtr*.

4.11.29 dorm22

```
DORM22 overwrites the general real M-by-N matrix C with

                 SIDE = 'L'     SIDE = 'R'
 TRANS = 'N':      Q * C          C * Q
 TRANS = 'T':      Q**T * C       C * Q**T

 where Q is a real orthogonal matrix of order NQ, with NQ = M if
 SIDE = 'L' and NQ = N if SIDE = 'R'.
 The orthogonal matrix Q processes a 2-by-2 block structure

        [  Q11  Q12  ]
    Q = [            ]
        [  Q21  Q22  ],

 where Q12 is an N1-by-N1 lower triangular matrix and Q21 is an
 N2-by-N2 upper triangular matrix.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dorm22(SIDE, TRANS, M, N, N1, N2, Q, LDQ, C, LDC, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dorm22_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *n1,
             const armpl_int_t *n2, const double *q, const armpl_int_t *ldq,
             double *c, const armpl_int_t *ldc, double *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply Q (No transpose); = 'C': apply Q^T (Conjugate transpose).

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

N1 Input parameter.

N2 Input parameter.

N1 is INTEGER

N2 is INTEGER The dimension of Q12 and Q21, respectively. N1, N2 >= 0. The following requirement must be satisfied: N1 + N2 = M if SIDE = 'L' and N1 + N2 = N if SIDE = 'R'.

Q Input parameter.

Q is DOUBLE PRECISION

Q is an array, dimension. (LDQ, M) if SIDE = 'L' (LDQ, N) if SIDE = 'R'

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, M) if SIDE = 'L'; LDQ >= max(1, N) if SIDE = 'R'.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, N); if SIDE = 'R', LWORK >= max(1, M). For optimum performance LWORK >= M*N.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *sorm22*.

4.11.30 dormtr

dormtr overwrites the general real M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'T': Q^T * C C * Q^T

where Q is a real orthogonal matrix of order nq, with nq = m if SIDE = 'L' and nq = n if SIDE = 'R'. Q is defined as the product of nq-1 elementary reflectors, as returned by DSYTRD:

if UPLO = 'U', Q = H(nq-1) . . . H(2) H(1);

if UPLO = 'L', Q = H(1) H(2) . . . H(nq-1).

Syntax

Fortran specification:

```
use armpl_library

subroutine dormtr(SIDE, UPLO, TRANS, M, N, A, LDA, TAU, C, LDC, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dormtr_(const char *side, const char *uplo, const char *trans,
             const armpl_int_t *m, const armpl_int_t *n, const double *a,
             const armpl_int_t *lda, const double *tau, double *c,
             const armpl_int_t *ldc, double *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A contains elementary reflectors from DSYTRD; = 'L': Lower triangle of A contains elementary reflectors from DSYTRD.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'T': Transpose, apply Q^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension. (LDA, M) if SIDE = 'L' (LDA, N) if SIDE = 'R' The vectors which define the elementary reflectors, as returned by DSYTRD.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M) if SIDE = 'L'; LDA >= max(1, N) if SIDE = 'R'.

TAU Input parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension. (M-1) if SIDE = 'L' (N-1) if SIDE = 'R' TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by DSYTRD.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, N); if SIDE = 'R', LWORK >= max(1, M). For optimum performance LWORK >= N*NB if SIDE = 'L', and LWORK >= M*NB if SIDE = 'R', where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *sormtr*. It also exists with a native C interface as *LAPACKE_dormtr*.

4.11.31 dpteqr

dpteqr computes all eigenvalues and, optionally, eigenvectors of a symmetric positive definite tridiagonal matrix by first factoring the matrix using DPTTRF, and then calling DBDSQR to compute the singular values of the bidiagonal factor.

This routine computes the eigenvalues of the positive definite tridiagonal matrix to high relative accuracy. This means that if the eigenvalues range over many orders of magnitude in size, then the small eigenvalues and corresponding eigenvectors will be computed more accurately than, for example, with the standard QR method.

The eigenvectors of a full or band symmetric positive definite matrix can also be found if DSYTRD, DSPTRD, or DSBTRD has been used to reduce this matrix to tridiagonal form. (The reduction to tridiagonal form, however,

may preclude the possibility of obtaining high relative accuracy in the small eigenvalues of the original matrix, if these eigenvalues range over many orders of magnitude.)

Syntax

Fortran specification:

```
use armpl_library

subroutine dpteqr(COMPZ, N, D, E, Z, LDZ, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dpteqr_(const char *compz, const armpl_int_t *n, double *d, double *e,
             double *z, const armpl_int_t *ldz, double *work,
             armpl_int_t *info, ... );
```

Parameters

COMPZ Input parameter.

COMPZ is CHARACTER*1

= 'N': Compute eigenvalues only. = 'V': Compute eigenvectors of original symmetric matrix also. Array Z contains the orthogonal matrix used to reduce the original matrix to tridiagonal form. = 'I': Compute eigenvectors of tridiagonal matrix also.

N Input parameter.

N is INTEGER

The order of the matrix. N >= 0.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry, the n diagonal elements of the tridiagonal matrix. On normal exit, D contains the eigenvalues, in descending order.

E Input and output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N-1). On entry, the (n-1) subdiagonal elements of the tridiagonal matrix. On exit, E has been destroyed.

Z Input and output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, N). On entry, if COMPZ = 'V', the orthogonal matrix used in the reduction to tridiagonal form. On exit, if COMPZ = 'V', the orthonormal eigenvectors of the original symmetric matrix; if COMPZ = 'I', the orthonormal eigenvectors of the tridiagonal matrix. If INFO > 0 on exit, Z contains the eigenvectors associated with only the stored eigenvalues. If COMPZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if COMPZ = 'V' or 'I', LDZ >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (4*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, and i is: <= N the Cholesky factorization of the matrix could not be performed because the i-th principal minor was not positive definite. > N the SVD algorithm failed to converge; if INFO = N+i, i off-diagonal elements of the bidiagonal factor did not converge to zero.

Related Information

For this routine in other precisions, please see *cpteqr*, *spteqr* and *zpteqr*. It also exists with a native C interface as *LAPACKE_dpteqr*.

4.11.32 dsbev

dsbev computes all the eigenvalues and, optionally, eigenvectors of a real symmetric band matrix A.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsbev(JOBZ, UPLO, N, KD, AB, LDAB, W, Z, LDZ, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dsbev_(const char *jobz, const char *uplo, const armpl_int_t *n,
            const armpl_int_t *kd, double *ab, const armpl_int_t *ldab,
            double *w, double *z, const armpl_int_t *ldz, double *work,
            armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, AB is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the first superdiagonal and the diagonal of the tridiagonal matrix T are returned in rows KD and KD+1 of AB, and if UPLO = 'L', the diagonal and first subdiagonal of T are returned in the first two rows of AB.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD + 1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the orthonormal eigenvectors of the matrix A, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (max(1,3*N-2)) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *ssbev*. It also exists with a native C interface as *LAPACKE_dsbev*.

4.11.33 dsbevd

dsbevd computes all the eigenvalues and, optionally, eigenvectors of a real symmetric band matrix A. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsbevd(JOBZ, UPLO, N, KD, AB, LDAB, W, Z, LDZ, WORK, LWORK, IWORK,
                  LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dsbevd_(const char *jobz, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *kd, double *ab, const armpl_int_t *ldab,
             double *w, double *z, const armpl_int_t *ldz, double *work,
             const armpl_int_t *lwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, AB is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the first superdiagonal and the diagonal of the tridiagonal matrix T are returned in rows KD and KD+1 of AB, and if UPLO = 'L', the diagonal and first subdiagonal of T are returned in the first two rows of AB.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD + 1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the orthonormal eigenvectors of the matrix A, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array,. dimension (LWORK) On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. IF N <= 1, LWORK must be at least 1. If JOBZ = 'N' and N > 2, LWORK must be at least 2*N. If JOBZ = 'V' and N > 2, LWORK must be at least (1 + 5* N + 2*N**2).

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. If JOBZ = 'N' or N <= 1, LIWORK must be at least 1. If JOBZ = 'V' and N > 2, LIWORK must be at least 3 + 5*N.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *ssbevd*. It also exists with a native C interface as *LAPACKE_dsbevd*.

4.11.34 dsbevx

dsbevx computes selected eigenvalues and, optionally, eigenvectors of a real symmetric band matrix A. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsbevx(JOBZ, RANGE, UPLO, N, KD, AB, LDAB, Q, LDQ, VL, VU, IL, IU,
                  ABSTOL, M, W, Z, LDZ, WORK, IWORK, IFAIL, INFO)
```

C specification:

```
#include "armpl.h"

void dsbevx_(const char *jobz, const char *range, const char *uplo,
             const armpl_int_t *n, const armpl_int_t *kd, double *ab,
             const armpl_int_t *ldab, double *q, const armpl_int_t *ldq,
             const double *vl, const double *vu, const armpl_int_t *il,
             const armpl_int_t *iu, const double *abstol, armpl_int_t *m,
             double *w, double *z, const armpl_int_t *ldz, double *work,
             armpl_int_t *iwork, armpl_int_t *ifail, armpl_int_t *info,
             ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found; = 'V': all eigenvalues in the half-open interval (VL,VU] will be found; = 'I': the IL-th through IU-th eigenvalues will be found.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, AB is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the first superdiagonal and the diagonal of the tridiagonal matrix T are returned in rows KD and KD+1 of AB, and if UPLO = 'L', the diagonal and first subdiagonal of T are returned in the first two rows of AB.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD + 1.

Q Output parameter.

Q is DOUBLE PRECISION

Q is an array, dimension (LDQ, N). If JOBZ = 'V', the N-by-N orthogonal matrix used in the reduction to tridiagonal form. If JOBZ = 'N', the array Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. If JOBZ = 'V', then LDQ >= max(1, N).

VL Input parameter.

VL is DOUBLE PRECISION

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is DOUBLE PRECISION

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is DOUBLE PRECISION

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing AB to tridiagonal form.

Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*DLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*DLAMCH('S').

See "Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy," by Demmel and Kahan, LAPACK Working Note #3.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). The first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). If an eigenvector fails to converge, then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL. If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (7*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (5*N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (N)

If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, then i eigenvectors failed to converge. Their indices are stored in array IFAIL.

Related Information

For this routine in other precisions, please see *ssbevx*. It also exists with a native C interface as *LAPACKE_dsbevx*.

4.11.35 dsbtrd

dsbtrd reduces a real symmetric band matrix A to symmetric tridiagonal form T by an orthogonal similarity transformation: Q^T * A * Q = T.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsbtrd(VECT, UPLO, N, KD, AB, LDAB, D, E, Q, LDQ, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dsbtrd_(const char *vect, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *kd, double *ab, const armpl_int_t *ldab,
             double *d, double *e, double *q, const armpl_int_t *ldq,
             double *work, armpl_int_t *info, ... );
```

Parameters

VECT Input parameter.

VECT is CHARACTER*1

= 'N': do not form Q; = 'V': form Q; = 'U': update a matrix X, by forming X*Q.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd). On exit, the diagonal elements of AB are overwritten by the diagonal elements of the tridiagonal matrix T; if KD > 0, the elements on the first superdiagonal (if UPLO = 'U') or the first subdiagonal (if UPLO = 'L') are overwritten by the off-diagonal elements of T; the rest of AB is overwritten by values generated during the reduction.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

D Output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The diagonal elements of the tridiagonal matrix T.

E Output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N-1). The off-diagonal elements of the tridiagonal matrix T: E(i) = T(i,i+1) if UPLO = 'U'; E(i) = T(i+1,i) if UPLO = 'L'.

Q Input and output parameter.

Q is DOUBLE PRECISION

Q is an array, dimension (LDQ, N). On entry, if VECT = 'U', then Q must contain an N-by-N matrix X; if VECT = 'N' or 'V', then Q need not be set.

On exit: if VECT = 'V', Q contains the N-by-N orthogonal matrix Q; if VECT = 'U', Q contains the product X*Q; if VECT = 'N', the array Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= 1, and LDQ >= N if VECT = 'V' or 'U'.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ssbtrd*. It also exists with a native C interface as *LAPACKE_dsbtrd*.

4.11.36 dspev

dspev computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix A in packed storage.

Syntax

Fortran specification:

```
use armpl_library

subroutine dspev(JOBZ, UPLO, N, AP, W, Z, LDZ, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dspev_(const char *jobz, const char *uplo, const armpl_int_t *n,
            double *ap, double *w, double *z, const armpl_int_t *ldz,
            double *work, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is DOUBLE PRECISION

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n.

On exit, AP is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the diagonal and first superdiagonal of the tridiagonal matrix T overwrite the corresponding elements of A, and if UPLO = 'L', the diagonal and first subdiagonal of T overwrite the corresponding elements of A.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the orthonormal eigenvectors of the matrix A, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (3*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *sspev*. It also exists with a native C interface as *LAPACKE_dspev*.

4.11.37 dspevd

dspevd computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix A in packed storage. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine dspevd(JOBZ, UPLO, N, AP, W, Z, LDZ, WORK, LWORK, IWORK, LIWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dspevd_(const char *jobz, const char *uplo, const armpl_int_t *n,
             double *ap, double *w, double *z, const armpl_int_t *ldz,
             double *work, const armpl_int_t *lwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is DOUBLE PRECISION

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n.

On exit, AP is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the diagonal and first superdiagonal of the tridiagonal matrix T overwrite the corresponding elements of A, and if UPLO = 'L', the diagonal and first subdiagonal of T overwrite the corresponding elements of A.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the orthonormal eigenvectors of the matrix A, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the required LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If N <= 1, LWORK must be at least 1. If JOBZ = 'N' and N > 1, LWORK must be at least 2*N. If JOBZ = 'V' and N > 1, LWORK must be at least 1 + 6* N + N**2.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the required sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the required LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. If JOBZ = 'N' or N <= 1, LIWORK must be at least 1. If JOBZ = 'V' and N > 1, LIWORK must be at least 3 + 5*N.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the required sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *sspevd*. It also exists with a native C interface as *LAPACKE_dspevd*.

4.11.38 dspevx

dspevx computes selected eigenvalues and, optionally, eigenvectors of a real symmetric matrix A in packed storage. Eigenvalues/vectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine dspevx(JOBZ, RANGE, UPLO, N, AP, VL, VU, IL, IU, ABSTOL, M, W, Z,
                  LDZ, WORK, IWORK, IFAIL, INFO)
```

C specification:

```
#include "armpl.h"

void dspevx_(const char *jobz, const char *range, const char *uplo,
             const armpl_int_t *n, double *ap, const double *vl,
             const double *vu, const armpl_int_t *il, const armpl_int_t *iu,
             const double *abstol, armpl_int_t *m, double *w, double *z,
             const armpl_int_t *ldz, double *work, armpl_int_t *iwork,
             armpl_int_t *ifail, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found; = 'V': all eigenvalues in the half-open interval (VL,VU] will be found; = 'I': the IL-th through IU-th eigenvalues will be found.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is DOUBLE PRECISION

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n.

On exit, AP is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the diagonal and first superdiagonal of the tridiagonal matrix T overwrite the corresponding elements of A, and if UPLO = 'L', the diagonal and first subdiagonal of T overwrite the corresponding elements of A.

VL Input parameter.

VL is DOUBLE PRECISION

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is DOUBLE PRECISION

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is DOUBLE PRECISION

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing AP to tridiagonal form.

Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*DLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*DLAMCH('S').

See "Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy," by Demmel and Kahan, LAPACK Working Note #3.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the selected eigenvalues in ascending order.

Z Output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). If an eigenvector fails to converge, then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL. If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (8*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (5*N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (N)

If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, then i eigenvectors failed to converge. Their indices are stored in array IFAIL.

Related Information

For this routine in other precisions, please see *sspevx*. It also exists with a native C interface as *LAPACKE_dspevx*.

4.11.39 dsptrd

dsptrd reduces a real symmetric matrix A stored in packed form to symmetric tridiagonal form T by an orthogonal similarity transformation: $Q^T * A * Q = T$.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsptrd(UPLO, N, AP, D, E, TAU, INFO)
```

C specification:

```
#include "armpl.h"

void dsptrd_(const char *uplo, const armpl_int_t *n, double *ap, double *d,
             double *e, double *tau, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is DOUBLE PRECISION

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n. On exit, if UPLO = 'U', the diagonal and first superdiagonal of A are overwritten by the corresponding elements of the tridiagonal matrix T, and the elements above the first superdiagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors; if UPLO = 'L', the diagonal and first subdiagonal of A are over- written by the corresponding elements of the tridiagonal matrix T, and the elements below the first subdiagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors. See Further Details.

D Output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The diagonal elements of the tridiagonal matrix T: D(i) = A(i,i).

E Output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N-1). The off-diagonal elements of the tridiagonal matrix T: E(i) = A(i,i+1) if UPLO = 'U', E(i) = A(i+1,i) if UPLO = 'L'.

TAU Output parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (N-1). The scalar factors of the elementary reflectors (see Further Details).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ssptrd*. It also exists with a native C interface as *LAPACKE_dsptrd*.

4.11.40 dstebz

dstebz computes the eigenvalues of a symmetric tridiagonal matrix T. The user may ask for all eigenvalues, all eigenvalues in the half-open interval (VL, VU], or the IL-th through IU-th eigenvalues.

To avoid overflow, the matrix must be scaled so that its largest element is no greater than overflow**(1/2) * underflow**(1/4) in absolute value, and for greatest accuracy, it should not be much smaller than that.

See W. Kahan "Accurate Eigenvalues of a Symmetric Tridiagonal Matrix", Report CS41, Computer Science Dept., Stanford University, July 21, 1966.

Syntax

Fortran specification:

```
use armpl_library

subroutine dstebz(RANGE, ORDER, N, VL, VU, IL, IU, ABSTOL, D, E, M, NSPLIT, W,
                  IBLOCK, ISPLIT, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dstebz_(const char *range, const char *order, const armpl_int_t *n,
             const double *vl, const double *vu, const armpl_int_t *il,
             const armpl_int_t *iu, const double *abstol, const double *d,
             const double *e, armpl_int_t *m, armpl_int_t *nsplit, double *w,
             armpl_int_t *iblock, armpl_int_t *isplit, double *work,
             armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': ("All") all eigenvalues will be found. = 'V': ("Value") all eigenvalues in the half-open interval (VL, VU] will be found. = 'I': ("Index") the IL-th through IU-th eigenvalues (of the entire matrix) will be found.

ORDER Input parameter.

ORDER is CHARACTER*1

= 'B': ("By Block") the eigenvalues will be grouped by split-off block (see IBLOCK, ISPLIT) and ordered from smallest to largest within the block. = 'E': ("Entire matrix") the eigenvalues for the entire matrix will be ordered from smallest to largest.

N Input parameter.

N is INTEGER

The order of the tridiagonal matrix T. N >= 0.

VL Input parameter.

VL is DOUBLE PRECISION

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. Eigenvalues less than or equal to VL, or greater than VU, will not be returned. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is DOUBLE PRECISION

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. Eigenvalues less than or equal to VL, or greater than VU, will not be returned. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is DOUBLE PRECISION

The absolute tolerance for the eigenvalues. An eigenvalue (or cluster) is considered to be located if it has been determined to lie in an interval whose width is ABSTOL or less. If ABSTOL is less than or equal to zero, then ULP*|T| will be used, where |T| means the 1-norm of T.

Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*DLAMCH('S'), not zero.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The n diagonal elements of the tridiagonal matrix T.

E Input parameter.

E is DOUBLE PRECISION

E is an array, dimension (N-1). The (n-1) off-diagonal elements of the tridiagonal matrix T.

M Output parameter.

M is INTEGER

The actual number of eigenvalues found. 0 <= M <= N. (See also the description of INFO=2,3.)

NSPLIT Output parameter.

NSPLIT is INTEGER

The number of diagonal blocks in the matrix T. 1 <= NSPLIT <= N.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). On exit, the first M elements of W will contain the eigenvalues. (DSTEBZ may use the remaining N-M elements as workspace.)

IBLOCK Output parameter.

IBLOCK is INTEGER array, dimension (N)

At each row/column j where E(j) is zero or small, the matrix T is considered to split into a block diagonal matrix. On exit, if INFO = 0, IBLOCK(i) specifies to which block (from 1 to the number of blocks) the eigenvalue W(i) belongs. (DSTEBZ may use the remaining N-M elements as workspace.)

ISPLIT Output parameter.

ISPLIT is INTEGER array, dimension (N)

The splitting points, at which T breaks up into submatrices. The first submatrix consists of rows/columns 1 to ISPLIT(1), the second of rows/columns ISPLIT(1)+1 through ISPLIT(2), etc., and the NSPLIT-th consists of rows/columns ISPLIT(NSPLIT-1)+1 through ISPLIT(NSPLIT)=N. (Only the first NSPLIT elements will actually be used, but since the user cannot know a priori what value NSPLIT will have, N words must be reserved for ISPLIT.)

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (4*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (3*N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: some or all of the eigenvalues failed to converge or were not computed: =1 or 3: Bisection failed to converge for some eigenvalues; these eigenvalues are flagged by a negative block number. The effect is that the eigenvalues may not be as accurate as the absolute and relative tolerances. This is generally caused by unexpectedly inaccurate arithmetic. =2 or 3: RANGE='I' only: Not all of the eigenvalues IL:IU were found. Effect: M < IU+1-IL Cause: non-monotonic arithmetic, causing the Sturm sequence to be non-monotonic. Cure: recalculate, using RANGE='A', and pick out eigenvalues IL:IU. In some cases, increasing the PARAMETER "FUDGE" may make things work. = 4: RANGE='I', and the Gershgorin interval initially used was too small. No eigenvalues were computed. Probable cause: your machine has sloppy floating-point arithmetic. Cure: Increase the PARAMETER "FUDGE", recompile, and try again.

Related Information

For this routine in other precisions, please see *sstebz*. It also exists with a native C interface as *LAPACKE_dstebz*.

4.11.41 dstedc

`dstedc` computes all eigenvalues and, optionally, eigenvectors of a symmetric tridiagonal matrix using the divide and conquer method. The eigenvectors of a full or band real symmetric matrix can also be found if DSYTRD or DSPTRD or DSBTRD has been used to reduce this matrix to tridiagonal form.

This code makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none. See DLAED3 for details.

Syntax

Fortran specification:

```
use armpl_library

subroutine dstedc(COMPZ, N, D, E, Z, LDZ, WORK, LWORK, IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dstedc_(const char *compz, const armpl_int_t *n, double *d, double *e,
             double *z, const armpl_int_t *ldz, double *work,
             const armpl_int_t *lwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

COMPZ Input parameter.

`COMPZ` is `CHARACTER*1`

= 'N': Compute eigenvalues only. = 'I': Compute eigenvectors of tridiagonal matrix also. = 'V': Compute eigenvectors of original dense symmetric matrix also. On entry, `Z` contains the orthogonal matrix used to reduce the original matrix to tridiagonal form.

N Input parameter.

`N` is `INTEGER`

The dimension of the symmetric tridiagonal matrix. `N` >= 0.

D Input and output parameter.

`D` is `DOUBLE PRECISION`

`D` is an array, dimension (`N`). On entry, the diagonal elements of the tridiagonal matrix. On exit, if `INFO` = 0, the eigenvalues in ascending order.

E Input and output parameter.

`E` is `DOUBLE PRECISION`

`E` is an array, dimension (N-1). On entry, the subdiagonal elements of the tridiagonal matrix. On exit, `E` has been destroyed.

Z Input and output parameter.

`Z` is `DOUBLE PRECISION`

`Z` is an array, dimension (`LDZ`, `N`). On entry, if `COMPZ` = 'V', then `Z` contains the orthogonal matrix used in the reduction to tridiagonal form. On exit, if `INFO` = 0, then if `COMPZ` = 'V', `Z` contains the orthonormal

eigenvectors of the original symmetric matrix, and if COMPZ = 'I', Z contains the orthonormal eigenvectors of the symmetric tridiagonal matrix. If COMPZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1. If eigenvectors are desired, then LDZ >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If COMPZ = 'N' or N <= 1 then LWORK must be at least 1. If COMPZ = 'V' and N > 1 then LWORK must be at least (1 + 3* N + 2*N*lg N + 4*N**2), where lg(N) = smallest integer k such that 2**k >= N. If COMPZ = 'I' and N > 1 then LWORK must be at least (1 + 4* N + N**2). Note that for COMPZ = 'I' or 'V', then if N is less than or equal to the minimum divide size, usually 25, then LWORK need only be max(1,2*(N-1)).

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. If COMPZ = 'N' or N <= 1 then LIWORK must be at least 1. If COMPZ = 'V' and N > 1 then LIWORK must be at least (6 + 6* N + 5*N*lg N). If COMPZ = 'I' and N > 1 then LIWORK must be at least (3 + 5* N). Note that for COMPZ = 'I' or 'V', then if N is less than or equal to the minimum divide size, usually 25, then LIWORK need only be 1.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the IWORK array, returns this value as the first entry of the IWORK array, and no error message related to LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: The algorithm failed to compute an eigenvalue while working on the submatrix lying in rows and columns INFO/(N+1) through mod(INFO,N+1).

Related Information

For this routine in other precisions, please see *cstedc*, *sstedc* and *zstedc*. It also exists with a native C interface as *LAPACKE_dstedc*.

4.11.42 dstegr

dstegr computes selected eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix T. Any such unreduced matrix has a well defined set of pairwise different real eigenvalues, the corresponding real eigenvectors are pairwise orthogonal.

The spectrum may be computed either completely or partially by specifying either an interval (VL,VU] or a range of indices IL:IU for the desired eigenvalues.

dstegr is a compatibility wrapper around the improved DSTEMR routine. See DSTEMR for further details.

One important change is that the ABSTOL parameter no longer provides any benefit and hence is no longer used.

Note : dstegr and DSTEMR work only on machines which follow IEEE-754 floating-point standard in their handling of infinities and NaNs. Normal execution may create these exceptiona values and hence may abort due to a floating point exception in environments which do not conform to the IEEE-754 standard.

Syntax

Fortran specification:

```
use armpl_library

subroutine dstegr(JOBZ, RANGE, N, D, E, VL, VU, IL, IU, ABSTOL, M, W, Z, LDZ,
                  ISUPPZ, WORK, LWORK, IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dstegr_(const char *jobz, const char *range, const armpl_int_t *n,
             double *d, double *e, const double *vl, const double *vu,
             const armpl_int_t *il, const armpl_int_t *iu,
             const double *abstol, armpl_int_t *m, double *w, double *z,
             const armpl_int_t *ldz, armpl_int_t *isuppz, double *work,
             const armpl_int_t *lwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found. = 'V': all eigenvalues in the half-open interval (VL,VU] will be found. = 'I': the IL-th through IU-th eigenvalues will be found.

N Input parameter.

N is INTEGER

The order of the matrix. N >= 0.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry, the N diagonal elements of the tridiagonal matrix T. On exit, D is overwritten.

E Input and output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N). On entry, the (N-1) subdiagonal elements of the tridiagonal matrix T in elements 1 to N-1 of E. E(N) need not be set on input, but is used internally as workspace. On exit, E is overwritten.

VL Input parameter.

VL is DOUBLE PRECISION

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is DOUBLE PRECISION

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is DOUBLE PRECISION

Unused. Was the absolute error tolerance for the eigenvalues/eigenvectors in previous versions.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). The first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', and if INFO = 0, then the first M columns of Z contain the orthonormal eigenvectors of the matrix T corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used. Supplying N columns is always safe.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', then LDZ >= max(1, N).

ISUPPZ Output parameter.

ISUPPZ is INTEGER array, dimension (2*max(1, M))

The support of the eigenvectors in Z, i.e., the indices indicating the nonzero elements in Z. The i-th computed eigenvector is nonzero only in elements ISUPPZ(2*i-1) through ISUPPZ(2*i). This is relevant in the case when the matrix is split. ISUPPZ is only accessed when JOBZ is 'V' and N > 0.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (LWORK). On exit, if INFO = 0, WORK(1) returns the optimal (and minimal) LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,18*N) if JOBZ = 'V', and LWORK >= max(1,12*N) if JOBZ = 'N'. If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (LIWORK)

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. LIWORK >= max(1,10*N) if the eigenvectors are desired, and LIWORK >= max(1,8*N) if only the eigenvalues are to be computed. If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the IWORK array, returns this value as the first entry of the IWORK array, and no error message related to LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

On exit, INFO = 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = 1X, internal error in DLARRE, if INFO = 2X, internal error in DLARRV. Here, the digit X = ABS(IINFO) < 10, where IINFO is the nonzero error code returned by DLARRE or DLARRV, respectively.

Related Information

For this routine in other precisions, please see *cstegr*, *sstegr* and *zstegr*. It also exists with a native C interface as *LAPACKE_dstegr*.

4.11.43 dstein

dstein computes the eigenvectors of a real symmetric tridiagonal matrix T corresponding to specified eigenvalues, using inverse iteration.

The maximum number of iterations allowed for each eigenvector is specified by an internal parameter MAXITS (currently set to 5).

Syntax

Fortran specification:

```
use armpl_library

subroutine dstein(N, D, E, M, W, IBLOCK, ISPLIT, Z, LDZ, WORK, IWORK, IFAIL,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dstein_(const armpl_int_t *n, const double *d, const double *e,
             const armpl_int_t *m, const double *w, const armpl_int_t *iblock,
             const armpl_int_t *isplit, double *z, const armpl_int_t *ldz,
             double *work, armpl_int_t *iwork, armpl_int_t *ifail,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix. N >= 0.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The n diagonal elements of the tridiagonal matrix T.

E Input parameter.

E is DOUBLE PRECISION

E is an array, dimension (N-1). The (n-1) subdiagonal elements of the tridiagonal matrix T, in elements 1 to N-1.

M Input parameter.

M is INTEGER

The number of eigenvectors to be found. 0 <= M <= N.

W Input parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). The first M elements of W contain the eigenvalues for which eigenvectors are to be computed. The eigenvalues should be grouped by split-off block and ordered from smallest to largest within the block. (The output array W from DSTEBZ with ORDER = 'B' is expected here.)

IBLOCK Input parameter.

IBLOCK is INTEGER array, dimension (N)

The submatrix indices associated with the corresponding eigenvalues in W; IBLOCK(i)=1 if eigenvalue W(i) belongs to the first submatrix from the top, =2 if W(i) belongs to the second submatrix, etc. (The output array IBLOCK from DSTEBZ is expected here.)

ISPLIT Input parameter.

ISPLIT is INTEGER array, dimension (N)

The splitting points, at which T breaks up into submatrices. The first submatrix consists of rows/columns 1 to ISPLIT(1), the second of rows/columns ISPLIT(1)+1 through ISPLIT(2), etc. (The output array ISPLIT from DSTEBZ is expected here.)

Z Output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, M). The computed eigenvectors. The eigenvector associated with the eigenvalue W(i) is stored in the i-th column of Z. Any vector which fails to converge is set to its current iterate after MAXITS iterations.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (5*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (M)

On normal exit, all elements of IFAIL are zero. If one or more eigenvectors fail to converge after MAXITS iterations, then their indices are stored in array IFAIL.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, then i eigenvectors failed to converge in MAXITS iterations. Their indices are stored in array IFAIL.

Related Information

For this routine in other precisions, please see *cstein*, *sstein* and *zstein*. It also exists with a native C interface as *LAPACKE_dstein*.

4.11.44 dstemr

dstemr computes selected eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix T. Any such unreduced matrix has a well defined set of pairwise different real eigenvalues, the corresponding real eigenvectors are pairwise orthogonal.

The spectrum may be computed either completely or partially by specifying either an interval (VL,VU] or a range of indices IL:IU for the desired eigenvalues.

Depending on the number of desired eigenvalues, these are computed either by bisection or the dqds algorithm. Numerically orthogonal eigenvectors are computed by the use of various suitable L D L^T factorizations near clusters of close eigenvalues (referred to as RRRs, Relatively Robust Representations). An informal sketch of the algorithm follows.

For each unreduced block (submatrix) of T,

```
(a) Compute T - sigma I  = L D L^T, so that L and D
    define all the wanted eigenvalues to high relative accuracy.
    This means that small relative changes in the entries of D and L
    cause only small relative changes in the eigenvalues and
    eigenvectors. The standard (unfactored) representation of the
    tridiagonal matrix T does not have this property in general.
(b) Compute the eigenvalues to suitable accuracy.
    If the eigenvectors are desired, the algorithm attains full
    accuracy of the computed eigenvalues only right before
    the corresponding vectors have to be computed, see steps c) and d).
(c) For each cluster of close eigenvalues, select a new
    shift close to the cluster, find a new factorization, and refine
    the shifted eigenvalues to suitable accuracy.
(d) For each eigenvalue with a large enough relative separation compute
```

(continues on next page)

(continued from previous page)

```
    the corresponding eigenvector by forming a rank revealing twisted
    factorization. Go back to (c) for any clusters that remain.
```

For more details, see: - Inderjit S. Dhillon and Beresford N. Parlett: “Multiple representations

```
 to compute orthogonal eigenvectors of symmetric tridiagonal matrices,"
Linear Algebra and its Applications, 387(1), pp. 1-28, August 2004.
```

- Inderjit Dhillon and Beresford Parlett: “Orthogonal Eigenvectors and

```
 Relative Gaps," SIAM Journal on Matrix Analysis and Applications, Vol. 25,
2004.  Also LAPACK Working Note 154.
```

- Inderjit Dhillon: “A new O(n^2) algorithm for the symmetric

```
 tridiagonal eigenvalue/eigenvector problem",
Computer Science Division Technical Report No. UCB/CSD-97-971,
UC Berkeley, May 1997.
```

Further Details 1.‘‘dstemr‘‘ works only on machines which follow IEEE-754 floating-point standard in their handling of infinities and NaNs. This permits the use of efficient inner loops avoiding a check for zero divisors.

Syntax

Fortran specification:

```
use armpl_library

subroutine dstemr(JOBZ, RANGE, N, D, E, VL, VU, IL, IU, M, W, Z, LDZ, NZC,
                  ISUPPZ, TRYRAC, WORK, LWORK, IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dstemr_(const char *jobz, const char *range, const armpl_int_t *n,
             double *d, double *e, const double *vl, const double *vu,
             const armpl_int_t *il, const armpl_int_t *iu, armpl_int_t *m,
             double *w, double *z, const armpl_int_t *ldz,
             const armpl_int_t *nzc, armpl_int_t *isuppz, armpl_int_t *tryrac,
             double *work, const armpl_int_t *lwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= ‘N’: Compute eigenvalues only; = ‘V’: Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= ‘A’: all eigenvalues will be found. = ‘V’: all eigenvalues in the half-open interval (VL,VU] will be found. = ‘I’: the IL-th through IU-th eigenvalues will be found.

N Input parameter.

N is INTEGER

The order of the matrix. N >= 0.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry, the N diagonal elements of the tridiagonal matrix T. On exit, D is overwritten.

E Input and output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N). On entry, the (N-1) subdiagonal elements of the tridiagonal matrix T in elements 1 to N-1 of E. E(N) need not be set on input, but is used internally as workspace. On exit, E is overwritten.

VL Input parameter.

VL is DOUBLE PRECISION

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is DOUBLE PRECISION

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0. Not referenced if RANGE = 'A' or 'V'.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). The first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', and if INFO = 0, then the first M columns of Z contain the orthonormal eigenvectors of the matrix T corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and can be computed with a workspace query by setting NZC = -1, see below.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', then LDZ >= max(1, N).

NZC Input parameter.

NZC is INTEGER

The number of eigenvectors to be held in the array Z. If RANGE = 'A', then NZC >= max(1, N). If RANGE = 'V', then NZC >= the number of eigenvalues in (VL,VU]. If RANGE = 'I', then NZC >= IU-IL+1. If NZC = -1, then a workspace query is assumed; the routine calculates the number of columns of the array Z that are needed to hold the eigenvectors. This value is returned as the first entry of the Z array, and no error message related to NZC is issued by XERBLA.

ISUPPZ Output parameter.

ISUPPZ is INTEGER array, dimension (2*max(1, M))

The support of the eigenvectors in Z, i.e., the indices indicating the nonzero elements in Z. The i-th computed eigenvector is nonzero only in elements ISUPPZ(2*i-1) through ISUPPZ(2*i). This is relevant in the case when the matrix is split. ISUPPZ is only accessed when JOBZ is 'V' and N > 0.

TRYRAC Input and output parameter.

TRYRAC is LOGICAL

If TRYRAC.EQ..TRUE., indicates that the code should check whether the tridiagonal matrix defines its eigenvalues to high relative accuracy. If so, the code uses relative-accuracy preserving algorithms that might be (a bit) slower depending on the matrix. If the matrix does not define its eigenvalues to high relative accuracy, the code can uses possibly faster algorithms. If TRYRAC.EQ..FALSE., the code is not required to guarantee relatively accurate eigenvalues and can use the fastest possible techniques. On exit, a .TRUE. TRYRAC will be set to .FALSE. if the matrix does not define its eigenvalues to high relative accuracy.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (LWORK). On exit, if INFO = 0, WORK(1) returns the optimal (and minimal) LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,18*N) if JOBZ = 'V', and LWORK >= max(1,12*N) if JOBZ = 'N'. If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (LIWORK)

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. LIWORK >= max(1,10*N) if the eigenvectors are desired, and LIWORK >= max(1,8*N) if only the eigenvalues are to be computed. If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the IWORK array, returns this value as the first entry of the IWORK array, and no error message related to LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

On exit, INFO = 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = 1X, internal error in DLARRE, if INFO = 2X, internal error in DLARRV. Here, the digit X = ABS(IINFO) < 10, where IINFO is the nonzero error code returned by DLARRE or DLARRV, respectively.

Related Information

For this routine in other precisions, please see *cstemr*, *sstemr* and *zstemr*. It also exists with a native C interface as *LAPACKE_dstemr*.

4.11.45 dsteqr

dsteqr computes all eigenvalues and, optionally, eigenvectors of a symmetric tridiagonal matrix using the implicit QL or QR method. The eigenvectors of a full or band symmetric matrix can also be found if DSYTRD or DSPTRD or DSBTRD has been used to reduce this matrix to tridiagonal form.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsteqr(COMPZ, N, D, E, Z, LDZ, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dsteqr_(const char *compz, const armpl_int_t *n, double *d, double *e,
             double *z, const armpl_int_t *ldz, double *work,
             armpl_int_t *info, ... );
```

Parameters

COMPZ Input parameter.

COMPZ is CHARACTER*1

= 'N': Compute eigenvalues only. = 'V': Compute eigenvalues and eigenvectors of the original symmetric matrix. On entry, Z must contain the orthogonal matrix used to reduce the original matrix to tridiagonal form. = 'I': Compute eigenvalues and eigenvectors of the tridiagonal matrix. Z is initialized to the identity matrix.

N Input parameter.

N is INTEGER

The order of the matrix. N >= 0.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry, the diagonal elements of the tridiagonal matrix. On exit, if INFO = 0, the eigenvalues in ascending order.

E Input and output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N-1). On entry, the (n-1) subdiagonal elements of the tridiagonal matrix. On exit, E has been destroyed.

Z Input and output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, N). On entry, if COMPZ = 'V', then Z contains the orthogonal matrix used in the reduction to tridiagonal form. On exit, if INFO = 0, then if COMPZ = 'V', Z contains the orthonormal

eigenvectors of the original symmetric matrix, and if COMPZ = 'I', Z contains the orthonormal eigenvectors of the symmetric tridiagonal matrix. If COMPZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if eigenvectors are desired, then LDZ >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (max(1,2*N-2)). If COMPZ = 'N', then WORK is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: the algorithm has failed to find all the eigenvalues in a total of 30* N iterations; if INFO = i, then i elements of E have not converged to zero; on exit, D and E contain the elements of a symmetric tridiagonal matrix which is orthogonally similar to the original matrix.

Related Information

For this routine in other precisions, please see *csteqr*, *ssteqr* and *zsteqr*. It also exists with a native C interface as *LAPACKE_dsteqr*.

4.11.46 dsterf

dsterf computes all eigenvalues of a symmetric tridiagonal matrix using the Pal-Walker-Kahan variant of the QL or QR algorithm.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsterf(N, D, E, INFO)
```

C specification:

```
#include "armpl.h"

void dsterf_(const armpl_int_t *n, double *d, double *e, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix. N >= 0.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry, the n diagonal elements of the tridiagonal matrix. On exit, if INFO = 0, the eigenvalues in ascending order.

E Input and output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N-1). On entry, the (n-1) subdiagonal elements of the tridiagonal matrix. On exit, E has been destroyed.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: the algorithm failed to find all of the eigenvalues in a total of 30* N iterations; if INFO = i, then i elements of E have not converged to zero.

Related Information

For this routine in other precisions, please see *ssterf*. It also exists with a native C interface as *LAPACKE_dsterf*.

4.11.47 dstev

dstev computes all eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix A.

Syntax

Fortran specification:

```
use armpl_library

subroutine dstev(JOBZ, N, D, E, Z, LDZ, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dstev_(const char *jobz, const armpl_int_t *n, double *d, double *e,
            double *z, const armpl_int_t *ldz, double *work,
            armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

N Input parameter.

N is INTEGER

The order of the matrix. N >= 0.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry, the n diagonal elements of the tridiagonal matrix A. On exit, if INFO = 0, the eigenvalues in ascending order.

E Input and output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N-1). On entry, the (n-1) subdiagonal elements of the tridiagonal matrix A, stored in elements 1 to N-1 of E. On exit, the contents of E are destroyed.

Z Output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the orthonormal eigenvectors of the matrix A, with the i-th column of Z holding the eigenvector associated with D(i). If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (max(1,2*N-2)). If JOBZ = 'N', WORK is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of E did not converge to zero.

Related Information

For this routine in other precisions, please see *sstev*. It also exists with a native C interface as *LAPACKE_dstev*.

4.11.48 dstevd

dstevd computes all eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine dstevd(JOBZ, N, D, E, Z, LDZ, WORK, LWORK, IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dstevd_(const char *jobz, const armpl_int_t *n, double *d, double *e,
             double *z, const armpl_int_t *ldz, double *work,
             const armpl_int_t *lwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

N Input parameter.

N is INTEGER

The order of the matrix. N >= 0.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry, the n diagonal elements of the tridiagonal matrix A. On exit, if INFO = 0, the eigenvalues in ascending order.

E Input and output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N-1). On entry, the (n-1) subdiagonal elements of the tridiagonal matrix A, stored in elements 1 to N-1 of E. On exit, the contents of E are destroyed.

Z Output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the orthonormal eigenvectors of the matrix A, with the i-th column of Z holding the eigenvector associated with D(i). If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array,. dimension (LWORK) On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If JOBZ = 'N' or N <= 1 then LWORK must be at least 1. If JOBZ = 'V' and N > 1 then LWORK must be at least (1 + 4* N + N**2).

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. If JOBZ = 'N' or N <= 1 then LIWORK must be at least 1. If JOBZ = 'V' and N > 1 then LIWORK must be at least 3+5*N.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of E did not converge to zero.

Related Information

For this routine in other precisions, please see *sstevd*. It also exists with a native C interface as *LAPACKE_dstevd*.

4.11.49 dstevr

dstevr computes selected eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix T. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Whenever possible, dstevr calls DSTEMR to compute the eigenspectrum using Relatively Robust Representations. DSTEMR computes eigenvalues by the dqds algorithm, while orthogonal eigenvectors are computed from various "good" L D L^T representations (also known as Relatively Robust Representations). Gram-Schmidt orthogonalization is avoided as far as possible. More specifically, the various steps of the algorithm are as follows. For the i-th unreduced block of T,

```
(a) Compute T - sigma_i = L_i D_i L_i^T, such that L_i D_i L_i^T
     is a relatively robust representation,
(b) Compute the eigenvalues, lambda_j, of L_i D_i L_i^T to high
    relative accuracy by the dqds algorithm,
(c) If there is a cluster of close eigenvalues, "choose" sigma_i
    close to the cluster, and go to step (a),
(d) Given the approximate eigenvalue lambda_j of L_i D_i L_i^T,
    compute the corresponding eigenvector by forming a
    rank-revealing twisted factorization.
```

The desired accuracy of the output can be specified by the input parameter ABSTOL.

For more details, see "A new O(n^2) algorithm for the symmetric tridiagonal eigenvalue/eigenvector problem", by Inderjit Dhillon, Computer Science Division Technical Report No. UCB//CSD-97-971, UC Berkeley, May 1997.

Note 1 : dstevr calls DSTEMR when the full spectrum is requested on machines which conform to the ieee-754 floating point standard. dstevr calls DSTEBZ and DSTEIN on non-ieee machines and when partial spectrum requests are made.

Normal execution of DSTEMR may create NaNs and infinities and hence may abort due to a floating point exception in environments which do not handle NaNs and infinities in the ieee standard default manner.

Syntax

Fortran specification:

```
use armpl_library

subroutine dstevr(JOBZ, RANGE, N, D, E, VL, VU, IL, IU, ABSTOL, M, W, Z, LDZ,
                  ISUPPZ, WORK, LWORK, IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dstevr_(const char *jobz, const char *range, const armpl_int_t *n,
             double *d, double *e, const double *vl, const double *vu,
             const armpl_int_t *il, const armpl_int_t *iu,
             const double *abstol, armpl_int_t *m, double *w, double *z,
             const armpl_int_t *ldz, armpl_int_t *isuppz, double *work,
             const armpl_int_t *lwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found. = 'V': all eigenvalues in the half-open interval (VL,VU] will be found. = 'I': the IL-th through IU-th eigenvalues will be found. For RANGE = 'V' or 'I' and IU - IL < N - 1, DSTEBZ and DSTEIN are called

N Input parameter.

N is INTEGER

The order of the matrix. N >= 0.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry, the n diagonal elements of the tridiagonal matrix A. On exit, D may be multiplied by a constant factor chosen to avoid over/underflow in computing the eigenvalues.

E Input and output parameter.

E is DOUBLE PRECISION

E is an array, dimension (max(1,N-1)). On entry, the (n-1) subdiagonal elements of the tridiagonal matrix A in elements 1 to N-1 of E. On exit, E may be multiplied by a constant factor chosen to avoid over/underflow in computing the eigenvalues.

VL Input parameter.

VL is DOUBLE PRECISION

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is DOUBLE PRECISION

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is DOUBLE PRECISION

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing A to tridiagonal form.

See "Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy," by Demmel and Kahan, LAPACK Working Note #3.

If high relative accuracy is important, set ABSTOL to DLAMCH('Safe minimum'). Doing so will guarantee that eigenvalues are computed to high relative accuracy when possible in future releases. The current code does not make any guarantees about high relative accuracy, but future releases will. See J. Barlow and J. Demmel, "Computing Accurate Eigensystems of Scaled Diagonally Dominant Matrices", LAPACK Working Note #7, for a discussion of which matrices define their eigenvalues to high relative accuracy.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). The first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

ISUPPZ Output parameter.

ISUPPZ is INTEGER array, dimension (2*max(1, M))

The support of the eigenvectors in Z, i.e., the indices indicating the nonzero elements in Z. The i-th eigenvector is nonzero only in elements ISUPPZ(2*i-1) through ISUPPZ(2*i). Implemented only for RANGE = 'A' or 'I' and IU - IL = N - 1

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal (and minimal) LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,20*N).

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal (and minimal) LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. LIWORK >= max(1,10*N).

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: Internal error

Related Information

For this routine in other precisions, please see *sstevr*. It also exists with a native C interface as *LAPACKE_dstevr*.

4.11.50 dstevx

dstevx computes selected eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix A. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine dstevx(JOBZ, RANGE, N, D, E, VL, VU, IL, IU, ABSTOL, M, W, Z, LDZ,
                  WORK, IWORK, IFAIL, INFO)
```

C specification:

```
#include "armpl.h"

void dstevx_(const char *jobz, const char *range, const armpl_int_t *n,
             double *d, double *e, const double *vl, const double *vu,
             const armpl_int_t *il, const armpl_int_t *iu,
             const double *abstol, armpl_int_t *m, double *w, double *z,
             const armpl_int_t *ldz, double *work, armpl_int_t *iwork,
             armpl_int_t *ifail, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found. = 'V': all eigenvalues in the half-open interval (VL,VU] will be found. = 'I': the IL-th through IU-th eigenvalues will be found.

N Input parameter.

N is INTEGER

The order of the matrix. N >= 0.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry, the n diagonal elements of the tridiagonal matrix A. On exit, D may be multiplied by a constant factor chosen to avoid over/underflow in computing the eigenvalues.

E Input and output parameter.

E is DOUBLE PRECISION

E is an array, dimension (max(1,N-1)). On entry, the (n-1) subdiagonal elements of the tridiagonal matrix A in elements 1 to N-1 of E. On exit, E may be multiplied by a constant factor chosen to avoid over/underflow in computing the eigenvalues.

VL Input parameter.

VL is DOUBLE PRECISION

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is DOUBLE PRECISION

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is DOUBLE PRECISION

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix.

Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*DLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*DLAMCH('S').

See "Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy," by Demmel and Kahan, LAPACK Working Note #3.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). The first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). If an eigenvector fails to converge (INFO > 0), then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL. If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (5*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (5*N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (N)

If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, then i eigenvectors failed to converge. Their indices are stored in array IFAIL.

Related Information

For this routine in other precisions, please see *sstevx*. It also exists with a native C interface as *LAPACKE_dstevx*.

4.11.51 dsyev

`dsyev` computes all eigenvalues and, optionally, eigenvectors of a real symmetric matrix A.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsyev(JOBZ, UPLO, N, A, LDA, W, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dsyev_(const char *jobz, const char *uplo, const armpl_int_t *n,
            double *a, const armpl_int_t *lda, double *w, double *work,
            const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A. On exit, if JOBZ = 'V', then if INFO = 0, A contains the orthonormal eigenvectors of the matrix A. If JOBZ = 'N', then on exit the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= max(1,3*N-1). For optimal efficiency, LWORK >= (NB+2)*N, where NB is the blocksize for DSYTRD returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *ssyev*. It also exists with a native C interface as *LAPACKE_dsyev*.

4.11.52 dsyevd

dsyevd computes all eigenvalues and, optionally, eigenvectors of a real symmetric matrix A. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Because of large use of BLAS of level 3, dsyevd needs N**2 more workspace than DSYEVX.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsyevd(JOBZ, UPLO, N, A, LDA, W, WORK, LWORK, IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dsyevd_(const char *jobz, const char *uplo, const armpl_int_t *n,
             double *a, const armpl_int_t *lda, double *w, double *work,
             const armpl_int_t *lwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A. On exit, if JOBZ = 'V', then if INFO = 0, A contains the orthonormal eigenvectors of the matrix A. If JOBZ = 'N', then on exit the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array,. dimension (LWORK) On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If N <= 1, LWORK must be at least 1. If JOBZ = 'N' and N > 1, LWORK must be at least 2*N+1. If JOBZ = 'V' and N > 1, LWORK must be at least 1 + 6* N + 2*N**2.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. If N <= 1, LIWORK must be at least 1. If JOBZ = 'N' and N > 1, LIWORK must be at least 1. If JOBZ = 'V' and N > 1, LIWORK must be at least 3 + 5*N.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i and JOBZ = 'N', then the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not

converge to zero; if INFO = i and JOBZ = 'V', then the algorithm failed to compute an eigenvalue while working on the submatrix lying in rows and columns INFO/(N+1) through mod(INFO,N+1).

Related Information

For this routine in other precisions, please see *ssyevd*. It also exists with a native C interface as *LAPACKE_dsyevd*.

4.11.53 dsyevr

dsyevr computes selected eigenvalues and, optionally, eigenvectors of a real symmetric matrix A. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

dsyevr first reduces the matrix A to tridiagonal form T with a call to DSYTRD. Then, whenever possible, dsyevr calls DSTEMR to compute the eigenspectrum using Relatively Robust Representations. DSTEMR computes eigenvalues by the dqds algorithm, while orthogonal eigenvectors are computed from various "good" L D L^T representations (also known as Relatively Robust Representations). Gram-Schmidt orthogonalization is avoided as far as possible. More specifically, the various steps of the algorithm are as follows.

For each unreduced block (submatrix) of T,

```
 (a) Compute T - sigma I  = L D L^T, so that L and D
     define all the wanted eigenvalues to high relative accuracy.
     This means that small relative changes in the entries of D and L
     cause only small relative changes in the eigenvalues and
     eigenvectors. The standard (unfactored) representation of the
     tridiagonal matrix T does not have this property in general.
(b) Compute the eigenvalues to suitable accuracy.
     If the eigenvectors are desired, the algorithm attains full
     accuracy of the computed eigenvalues only right before
     the corresponding vectors have to be computed, see steps c) and d).
(c) For each cluster of close eigenvalues, select a new
     shift close to the cluster, find a new factorization, and refine
     the shifted eigenvalues to suitable accuracy.
(d) For each eigenvalue with a large enough relative separation compute
     the corresponding eigenvector by forming a rank revealing twisted
     factorization. Go back to (c) for any clusters that remain.
```

The desired accuracy of the output can be specified by the input parameter ABSTOL.

For more details, see DSTEMR's documentation and: - Inderjit S. Dhillon and Beresford N. Parlett: "Multiple representations

```
 to compute orthogonal eigenvectors of symmetric tridiagonal matrices,"
Linear Algebra and its Applications, 387(1), pp. 1-28, August 2004.
```

- Inderjit Dhillon and Beresford Parlett: "Orthogonal Eigenvectors and

```
 Relative Gaps," SIAM Journal on Matrix Analysis and Applications, Vol. 25,
2004.  Also LAPACK Working Note 154.
```

- Inderjit Dhillon: "A new O(n^2) algorithm for the symmetric

```
 tridiagonal eigenvalue/eigenvector problem",
Computer Science Division Technical Report No. UCB/CSD-97-971,
UC Berkeley, May 1997.
```

Note 1 : dsyevr calls DSTEMR when the full spectrum is requested on machines which conform to the ieee-754 floating point standard. dsyevr calls DSTEBZ and DSTEIN on non-ieee machines and when partial spectrum requests are made.

Normal execution of DSTEMR may create NaNs and infinities and hence may abort due to a floating point exception in environments which do not handle NaNs and infinities in the ieee standard default manner.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsyevr(JOBZ, RANGE, UPLO, N, A, LDA, VL, VU, IL, IU, ABSTOL, M, W,
                  Z, LDZ, ISUPPZ, WORK, LWORK, IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dsyevr_(const char *jobz, const char *range, const char *uplo,
             const armpl_int_t *n, double *a, const armpl_int_t *lda,
             const double *vl, const double *vu, const armpl_int_t *il,
             const armpl_int_t *iu, const double *abstol, armpl_int_t *m,
             double *w, double *z, const armpl_int_t *ldz,
             armpl_int_t *isuppz, double *work, const armpl_int_t *lwork,
             armpl_int_t *iwork, const armpl_int_t *liwork, armpl_int_t *info,
             ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found. = 'V': all eigenvalues in the half-open interval (VL,VU] will be found. = 'I': the IL-th through IU-th eigenvalues will be found. For RANGE = 'V' or 'I' and IU - IL < N - 1, DSTEBZ and DSTEIN are called

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A. On exit, the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

VL Input parameter.

VL is DOUBLE PRECISION

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is DOUBLE PRECISION

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is DOUBLE PRECISION

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing A to tridiagonal form.

See "Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy," by Demmel and Kahan, LAPACK Working Note #3.

If high relative accuracy is important, set ABSTOL to DLAMCH('Safe minimum'). Doing so will guarantee that eigenvalues are computed to high relative accuracy when possible in future releases. The current code does not make any guarantees about high relative accuracy, but future releases will. See J. Barlow and J. Demmel, "Computing Accurate Eigensystems of Scaled Diagonally Dominant Matrices", LAPACK Working Note #7, for a discussion of which matrices define their eigenvalues to high relative accuracy.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). The first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used. Supplying N columns is always safe.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

ISUPPZ Output parameter.

ISUPPZ is INTEGER array, dimension (2*max(1, M))

The support of the eigenvectors in Z, i.e., the indices indicating the nonzero elements in Z. The i-th eigenvector is nonzero only in elements ISUPPZ(2*i-1) through ISUPPZ(2*i). This is an output of DSTEMR (tridiagonal matrix). The support of the eigenvectors of A is typically 1:N because of the orthogonal transformations applied by DORMTR. Implemented only for RANGE = 'A' or 'I' and IU - IL = N - 1

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,26*N). For optimal efficiency, LWORK >= (NB+6)*N, where NB is the max of the blocksize for DSYTRD and DORMTR returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. LIWORK >= max(1,10*N).

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the IWORK array, returns this value as the first entry of the IWORK array, and no error message related to LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: Internal error

Related Information

For this routine in other precisions, please see *ssyevr*. It also exists with a native C interface as *LAPACKE_dsyevr*.

4.11.54 dsyevx

dsyevx computes selected eigenvalues and, optionally, eigenvectors of a real symmetric matrix A. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsyevx(JOBZ, RANGE, UPLO, N, A, LDA, VL, VU, IL, IU, ABSTOL, M, W,
                  Z, LDZ, WORK, LWORK, IWORK, IFAIL, INFO)
```

C specification:

```
#include "armpl.h"

void dsyevx_(const char *jobz, const char *range, const char *uplo,
             const armpl_int_t *n, double *a, const armpl_int_t *lda,
             const double *vl, const double *vu, const armpl_int_t *il,
             const armpl_int_t *iu, const double *abstol, armpl_int_t *m,
             double *w, double *z, const armpl_int_t *ldz, double *work,
             const armpl_int_t *lwork, armpl_int_t *iwork, armpl_int_t *ifail,
             armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found. = 'V': all eigenvalues in the half-open interval (VL,VU] will be found. = 'I': the IL-th through IU-th eigenvalues will be found.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A. On exit, the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

VL Input parameter.

VL is DOUBLE PRECISION

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is DOUBLE PRECISION

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is DOUBLE PRECISION

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing A to tridiagonal form.

Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*DLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*DLAMCH('S').

See "Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy," by Demmel and Kahan, LAPACK Working Note #3.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). On normal exit, the first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). If an eigenvector fails to converge, then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL. If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= 1, when N <= 1; otherwise 8*N. For optimal efficiency, LWORK >= (NB+3)*N, where NB is the max of the blocksize for DSYTRD and DORMTR returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (5*N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (N)

If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, then i eigenvectors failed to converge. Their indices are stored in array IFAIL.

Related Information

For this routine in other precisions, please see *ssyevx*. It also exists with a native C interface as *LAPACKE_dsyevx*.

4.11.55 dsytrd

dsytrd reduces a real symmetric matrix A to real symmetric tridiagonal form T by an orthogonal similarity transformation: $Q^T * A * Q = T$.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsytrd(UPLO, N, A, LDA, D, E, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dsytrd_(const char *uplo, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, double *d, double *e, double *tau,
             double *work, const armpl_int_t *lwork, armpl_int_t *info,
             ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, if UPLO = 'U', the diagonal and first superdiagonal of A are overwritten by the corresponding elements of the tridiagonal matrix T, and the elements above the first superdiagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors; if UPLO = 'L', the diagonal and first subdiagonal of A are over- written by the corresponding elements of the tridiagonal matrix T, and the elements below the first subdiagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

D Output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The diagonal elements of the tridiagonal matrix T: D(i) = A(i,i).

E Output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N-1). The off-diagonal elements of the tridiagonal matrix T: E(i) = A(i,i+1) if UPLO = 'U', E(i) = A(i+1,i) if UPLO = 'L'.

TAU Output parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (N-1). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= 1. For optimum performance LWORK >= N*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ssytrd*. It also exists with a native C interface as *LAPACKE_dsytrd*.

4.11.56 sdisna

sdisna computes the reciprocal condition numbers for the eigenvectors of a real symmetric or complex Hermitian matrix or for the left or right singular vectors of a general m-by-n matrix. The reciprocal condition number is the 'gap' between the corresponding eigenvalue or singular value and the nearest other one.

The bound on the error, measured by angle in radians, in the I-th computed vector is given by

```
SLAMCH( 'E' ) * ( ANORM / SEP( I ) )
```

where ANORM = 2-norm(A) = max(abs(D(j))). SEP(I) is not allowed to be smaller than SLAMCH('E')*ANORM in order to limit the size of the error bound.

sdisna may also be used to compute error bounds for eigenvectors of the generalized symmetric definite eigenproblem.

Syntax

Fortran specification:

```
use armpl_library

subroutine sdisna(JOB, M, N, D, SEP, INFO)
```

C specification:

```
#include "armpl.h"

void sdisna_(const char *job, const armpl_int_t *m, const armpl_int_t *n,
             const float *d, float *sep, armpl_int_t *info, ... );
```

Parameters

JOB Input parameter.

JOB is CHARACTER*1

Specifies for which problem the reciprocal condition numbers should be computed: = 'E': the eigenvectors of a symmetric/Hermitian matrix; = 'L': the left singular vectors of a general matrix; = 'R': the right singular vectors of a general matrix.

M Input parameter.

M is INTEGER

The number of rows of the matrix. M >= 0.

N Input parameter.

N is INTEGER

If JOB = 'L' or 'R', the number of columns of the matrix, in which case N >= 0. Ignored if JOB = 'E'.

D Input parameter.

D is REAL

D is an array, dimension (M) if JOB = 'E'. dimension (min(M, N)) if JOB = 'L' or 'R' The eigenvalues (if JOB = 'E') or singular values (if JOB = 'L' or 'R') of the matrix, in either increasing or decreasing order. If singular values, they must be non-negative.

SEP Output parameter.

SEP is REAL

SEP is an array, dimension (M) if JOB = 'E'. dimension (min(M, N)) if JOB = 'L' or 'R' The reciprocal condition numbers of the vectors.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *ddisna*. It also exists with a native C interface as *LAPACKE_sdisna*.

4.11.57 sopgtr

sopgtr generates a real orthogonal matrix Q which is defined as the product of n-1 elementary reflectors H(i) of order n, as returned by SSPTRD using packed storage:

if UPLO = 'U', Q = H(n-1) . . . H(2) H(1),

if UPLO = 'L', Q = H(1) H(2) . . . H(n-1).

Syntax

Fortran specification:

```
use armpl_library

subroutine sopgtr(UPLO, N, AP, TAU, Q, LDQ, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void sopgtr_(const char *uplo, const armpl_int_t *n, const float *ap,
             const float *tau, float *q, const armpl_int_t *ldq, float *work,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangular packed storage used in previous call to SSPTRD; = 'L': Lower triangular packed storage used in previous call to SSPTRD.

N Input parameter.

N is INTEGER

The order of the matrix Q. N >= 0.

AP Input parameter.

AP is REAL

AP is an array, dimension (N*(N+1)/2). The vectors which define the elementary reflectors, as returned by SSPTRD.

TAU Input parameter.

TAU is REAL

TAU is an array, dimension (N-1). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by SSPTRD.

Q Output parameter.

Q is REAL

Q is an array, dimension (LDQ, N). The N-by-N orthogonal matrix Q.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (N-1) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dopgtr*. It also exists with a native C interface as *LAPACKE_sopgtr*.

4.11.58 sopmtr

sopmtr overwrites the general real M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'T': Q^T * C C * Q^T

where Q is a real orthogonal matrix of order nq, with nq = m if SIDE = 'L' and nq = n if SIDE = 'R'. Q is defined as the product of nq-1 elementary reflectors, as returned by SSPTRD using packed storage:

if UPLO = 'U', Q = H(nq-1) . . . H(2) H(1);

if UPLO = 'L', Q = H(1) H(2) . . . H(nq-1).

Syntax

Fortran specification:

```
use armpl_library

subroutine sopmtr(SIDE, UPLO, TRANS, M, N, AP, TAU, C, LDC, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void sopmtr_(const char *side, const char *uplo, const char *trans,
             const armpl_int_t *m, const armpl_int_t *n, const float *ap,
             const float *tau, float *c, const armpl_int_t *ldc, float *work,
             armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangular packed storage used in previous call to SSPTRD; = 'L': Lower triangular packed storage used in previous call to SSPTRD.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'T': Transpose, apply Q^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

AP Input parameter.

AP is REAL

AP is an array, dimension. (M*(M+1)/2) if SIDE = 'L' (N*(N+1)/2) if SIDE = 'R' The vectors which define the elementary reflectors, as returned by SSPTRD. AP is modified by the routine but restored on exit.

TAU Input parameter.

TAU is REAL

TAU is an array, dimension (M-1) if SIDE = 'L'. or (N-1) if SIDE = 'R' TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by SSPTRD.

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension. (N) if SIDE = 'L' (M) if SIDE = 'R'

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dopmtr*. It also exists with a native C interface as *LAPACKE_sopmtr*.

4.11.59 sorgtr

sorgtr generates a real orthogonal matrix Q which is defined as the product of n-1 elementary reflectors of order N, as returned by SSYTRD:

if UPLO = 'U', Q = H(n-1) . . . H(2) H(1),

if UPLO = 'L', Q = H(1) H(2) . . . H(n-1).

Syntax

Fortran specification:

```
use armpl_library

subroutine sorgtr(UPLO, N, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sorgtr_(const char *uplo, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, const float *tau, float *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A contains elementary reflectors from SSYTRD; = 'L': Lower triangle of A contains elementary reflectors from SSYTRD.

N Input parameter.

N is INTEGER

The order of the matrix Q. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the vectors which define the elementary reflectors, as returned by SSYTRD. On exit, the N-by-N orthogonal matrix Q.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TAU Input parameter.

TAU is REAL

TAU is an array, dimension (N-1). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by SSYTRD.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,N-1). For optimum performance LWORK >= (N-1)*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dorgtr*. It also exists with a native C interface as *LAPACKE_sorgtr*.

4.11.60 sorm22

```
SORM22 overwrites the general real M-by-N matrix C with

                 SIDE = 'L'     SIDE = 'R'
 TRANS = 'N':      Q * C          C * Q
 TRANS = 'T':      Q**T * C       C * Q**T

 where Q is a real orthogonal matrix of order NQ, with NQ = M if
 SIDE = 'L' and NQ = N if SIDE = 'R'.
 The orthogonal matrix Q processes a 2-by-2 block structure

        [  Q11  Q12  ]
    Q = [            ]
        [  Q21  Q22  ],

 where Q12 is an N1-by-N1 lower triangular matrix and Q21 is an
 N2-by-N2 upper triangular matrix.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine sorm22(SIDE, TRANS, M, N, N1, N2, Q, LDQ, C, LDC, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void sorm22_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *n1,
             const armpl_int_t *n2, const float *q, const armpl_int_t *ldq,
             float *c, const armpl_int_t *ldc, float *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply Q (No transpose); = 'C': apply Q^T (Conjugate transpose).

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

N1 Input parameter.

N2 Input parameter.

N1 is INTEGER

N2 is INTEGER The dimension of Q12 and Q21, respectively. N1, N2 >= 0. The following requirement must be satisfied: N1 + N2 = M if SIDE = 'L' and N1 + N2 = N if SIDE = 'R'.

Q Input parameter.

Q is REAL

Q is an array, dimension. (LDQ, M) if SIDE = 'L' (LDQ, N) if SIDE = 'R'

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, M) if SIDE = 'L'; LDQ >= max(1, N) if SIDE = 'R'.

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, N); if SIDE = 'R', LWORK >= max(1, M). For optimum performance LWORK >= M*N.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dorm22*.

4.11.61 sormtr

sormtr overwrites the general real M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'T': Q^T * C C * Q^T

where Q is a real orthogonal matrix of order nq, with nq = m if SIDE = 'L' and nq = n if SIDE = 'R'. Q is defined as the product of nq-1 elementary reflectors, as returned by SSYTRD:

if UPLO = 'U', Q = H(nq-1) . . . H(2) H(1);

if UPLO = 'L', Q = H(1) H(2) . . . H(nq-1).

Syntax

Fortran specification:

```
use armpl_library

subroutine sormtr(SIDE, UPLO, TRANS, M, N, A, LDA, TAU, C, LDC, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void sormtr_(const char *side, const char *uplo, const char *trans,
             const armpl_int_t *m, const armpl_int_t *n, const float *a,
             const armpl_int_t *lda, const float *tau, float *c,
             const armpl_int_t *ldc, float *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A contains elementary reflectors from SSYTRD; = 'L': Lower triangle of A contains elementary reflectors from SSYTRD.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'T': Transpose, apply Q^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

A Input parameter.

A is REAL

A is an array, dimension. (LDA, M) if SIDE = 'L' (LDA, N) if SIDE = 'R' The vectors which define the elementary reflectors, as returned by SSYTRD.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M) if SIDE = 'L'; LDA >= max(1, N) if SIDE = 'R'.

TAU Input parameter.

TAU is REAL

TAU is an array, dimension. (M-1) if SIDE = 'L' (N-1) if SIDE = 'R' TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by SSYTRD.

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, N); if SIDE = 'R', LWORK >= max(1, M). For optimum performance LWORK >= N*NB if SIDE = 'L', and LWORK >= M*NB if SIDE = 'R', where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dormtr*. It also exists with a native C interface as *LAPACKE_sormtr*.

4.11.62 spteqr

spteqr computes all eigenvalues and, optionally, eigenvectors of a symmetric positive definite tridiagonal matrix by first factoring the matrix using SPTTRF, and then calling SBDSQR to compute the singular values of the bidiagonal factor.

This routine computes the eigenvalues of the positive definite tridiagonal matrix to high relative accuracy. This means that if the eigenvalues range over many orders of magnitude in size, then the small eigenvalues and corresponding eigenvectors will be computed more accurately than, for example, with the standard QR method.

The eigenvectors of a full or band symmetric positive definite matrix can also be found if SSYTRD, SSPTRD, or SSBTRD has been used to reduce this matrix to tridiagonal form. (The reduction to tridiagonal form, however, may preclude the possibility of obtaining high relative accuracy in the small eigenvalues of the original matrix, if these eigenvalues range over many orders of magnitude.)

Syntax

Fortran specification:

```
use armpl_library

subroutine spteqr(COMPZ, N, D, E, Z, LDZ, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void spteqr_(const char *compz, const armpl_int_t *n, float *d, float *e,
             float *z, const armpl_int_t *ldz, float *work, armpl_int_t *info,
             ... );
```

Parameters

COMPZ Input parameter.

COMPZ is CHARACTER*1

= 'N': Compute eigenvalues only. = 'V': Compute eigenvectors of original symmetric matrix also. Array Z contains the orthogonal matrix used to reduce the original matrix to tridiagonal form. = 'I': Compute eigenvectors of tridiagonal matrix also.

N Input parameter.

N is INTEGER

The order of the matrix. N >= 0.

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry, the n diagonal elements of the tridiagonal matrix. On normal exit, D contains the eigenvalues, in descending order.

E Input and output parameter.

E is REAL

E is an array, dimension (N-1). On entry, the (n-1) subdiagonal elements of the tridiagonal matrix. On exit, E has been destroyed.

Z Input and output parameter.

Z is REAL

Z is an array, dimension (LDZ, N). On entry, if COMPZ = 'V', the orthogonal matrix used in the reduction to tridiagonal form. On exit, if COMPZ = 'V', the orthonormal eigenvectors of the original symmetric matrix; if COMPZ = 'I', the orthonormal eigenvectors of the tridiagonal matrix. If INFO > 0 on exit, Z contains the eigenvectors associated with only the stored eigenvalues. If COMPZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if COMPZ = 'V' or 'I', LDZ >= max(1, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (4*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, and i is: <= N the Cholesky factorization of the matrix could not be performed because the i-th principal minor was not positive definite. > N the SVD algorithm failed to converge; if INFO = N+i, i off-diagonal elements of the bidiagonal factor did not converge to zero.

Related Information

For this routine in other precisions, please see *cpteqr*, *dpteqr* and *zpteqr*. It also exists with a native C interface as *LAPACKE_spteqr*.

4.11.63 ssbev

`ssbev` computes all the eigenvalues and, optionally, eigenvectors of a real symmetric band matrix A.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssbev(JOBZ, UPLO, N, KD, AB, LDAB, W, Z, LDZ, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void ssbev_(const char *jobz, const char *uplo, const armpl_int_t *n,
            const armpl_int_t *kd, float *ab, const armpl_int_t *ldab,
            float *w, float *z, const armpl_int_t *ldz, float *work,
            armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

`JOBZ` is `CHARACTER*1`

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

`N` is `INTEGER`

The order of the matrix A. `N` >= 0.

KD Input parameter.

`KD` is `INTEGER`

The number of superdiagonals of the matrix A if `UPLO` = 'U', or the number of subdiagonals if `UPLO` = 'L'. `KD` >= 0.

AB Input and output parameter.

`AB` is `REAL`

`AB` is an array, dimension (`LDAB`, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if `UPLO` = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if `UPLO` = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, AB is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the first superdiagonal and the diagonal of the tridiagonal matrix T are returned in rows KD and KD+1 of AB, and if UPLO = 'L', the diagonal and first subdiagonal of T are returned in the first two rows of AB.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD + 1.

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is REAL

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the orthonormal eigenvectors of the matrix A, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (max(1,3*N-2)) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *dsbev*. It also exists with a native C interface as *LAPACKE_ssbev*.

4.11.64 ssbevd

ssbevd computes all the eigenvalues and, optionally, eigenvectors of a real symmetric band matrix A. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssbevd(JOBZ, UPLO, N, KD, AB, LDAB, W, Z, LDZ, WORK, LWORK, IWORK,
                  LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ssbevd_(const char *jobz, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *kd, float *ab, const armpl_int_t *ldab,
             float *w, float *z, const armpl_int_t *ldz, float *work,
             const armpl_int_t *lwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is REAL

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, AB is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the first superdiagonal and the diagonal of the tridiagonal matrix T are returned in rows KD and KD+1 of AB, and if UPLO = 'L', the diagonal and first subdiagonal of T are returned in the first two rows of AB.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD + 1.

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is REAL

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the orthonormal eigenvectors of the matrix A, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is REAL

WORK is an array,. dimension (LWORK) On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. IF N <= 1, LWORK must be at least 1. If JOBZ = 'N' and N > 2, LWORK must be at least 2*N. If JOBZ = 'V' and N > 2, LWORK must be at least (1 + 5* N + 2*N**2).

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. If JOBZ = 'N' or N <= 1, LIWORK must be at least 1. If JOBZ = 'V' and N > 2, LIWORK must be at least 3 + 5*N.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *dsbevd*. It also exists with a native C interface as *LAPACKE_ssbevd*.

4.11.65 ssbevx

ssbevx computes selected eigenvalues and, optionally, eigenvectors of a real symmetric band matrix A. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssbevx(JOBZ, RANGE, UPLO, N, KD, AB, LDAB, Q, LDQ, VL, VU, IL, IU,
                  ABSTOL, M, W, Z, LDZ, WORK, IWORK, IFAIL, INFO)
```

C specification:

```
#include "armpl.h"

void ssbevx_(const char *jobz, const char *range, const char *uplo,
             const armpl_int_t *n, const armpl_int_t *kd, float *ab,
             const armpl_int_t *ldab, float *q, const armpl_int_t *ldq,
             const float *vl, const float *vu, const armpl_int_t *il,
             const armpl_int_t *iu, const float *abstol, armpl_int_t *m,
             float *w, float *z, const armpl_int_t *ldz, float *work,
             armpl_int_t *iwork, armpl_int_t *ifail, armpl_int_t *info,
             ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found; = 'V': all eigenvalues in the half-open interval (VL,VU] will be found; = 'I': the IL-th through IU-th eigenvalues will be found.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is REAL

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, AB is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the first superdiagonal and the diagonal of the tridiagonal matrix T are returned in rows KD and KD+1 of AB, and if UPLO = 'L', the diagonal and first subdiagonal of T are returned in the first two rows of AB.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD + 1.

Q Output parameter.

Q is REAL

Q is an array, dimension (LDQ, N). If JOBZ = 'V', the N-by-N orthogonal matrix used in the reduction to tridiagonal form. If JOBZ = 'N', the array Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. If JOBZ = 'V', then LDQ >= max(1, N).

VL Input parameter.

VL is REAL

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is REAL

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is REAL

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing AB to tridiagonal form.

Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*SLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*SLAMCH('S').

See "Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy," by Demmel and Kahan, LAPACK Working Note #3.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is REAL

W is an array, dimension (N). The first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is REAL

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). If an eigenvector fails to converge, then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL. If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (7*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (5*N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (N)

If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, then i eigenvectors failed to converge. Their indices are stored in array IFAIL.

Related Information

For this routine in other precisions, please see *dsbevx*. It also exists with a native C interface as *LAPACKE_ssbevx*.

4.11.66 ssbtrd

ssbtrd reduces a real symmetric band matrix A to symmetric tridiagonal form T by an orthogonal similarity transformation: $Q^T * A * Q = T$.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssbtrd(VECT, UPLO, N, KD, AB, LDAB, D, E, Q, LDQ, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void ssbtrd_(const char *vect, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *kd, float *ab, const armpl_int_t *ldab,
             float *d, float *e, float *q, const armpl_int_t *ldq,
             float *work, armpl_int_t *info, ... );
```

Parameters

VECT Input parameter.

VECT is CHARACTER*1

= 'N': do not form Q; = 'V': form Q; = 'U': update a matrix X, by forming X*Q.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is REAL

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd). On exit, the diagonal elements of AB are overwritten by the diagonal elements of the tridiagonal matrix T; if KD > 0, the elements on the first superdiagonal (if UPLO = 'U') or the first subdiagonal (if UPLO = 'L') are overwritten by the off-diagonal elements of T; the rest of AB is overwritten by values generated during the reduction.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

D Output parameter.

D is REAL

D is an array, dimension (N). The diagonal elements of the tridiagonal matrix T.

E Output parameter.

E is REAL

E is an array, dimension (N-1). The off-diagonal elements of the tridiagonal matrix T: E(i) = T(i,i+1) if UPLO = 'U'; E(i) = T(i+1,i) if UPLO = 'L'.

Q Input and output parameter.

Q is REAL

Q is an array, dimension (LDQ, N). On entry, if VECT = 'U', then Q must contain an N-by-N matrix X; if VECT = 'N' or 'V', then Q need not be set.

On exit: if VECT = 'V', Q contains the N-by-N orthogonal matrix Q; if VECT = 'U', Q contains the product X*Q; if VECT = 'N', the array Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= 1, and LDQ >= N if VECT = 'V' or 'U'.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dsbtrd*. It also exists with a native C interface as *LAPACKE_ssbtrd*.

4.11.67 sspev

sspev computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix A in packed storage.

Syntax

Fortran specification:

```
use armpl_library

subroutine sspev(JOBZ, UPLO, N, AP, W, Z, LDZ, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void sspev_(const char *jobz, const char *uplo, const armpl_int_t *n,
            float *ap, float *w, float *z, const armpl_int_t *ldz,
            float *work, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is REAL

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n.

On exit, AP is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the diagonal and first superdiagonal of the tridiagonal matrix T overwrite the corresponding elements of A, and if UPLO = 'L', the diagonal and first subdiagonal of T overwrite the corresponding elements of A.

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is REAL

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the orthonormal eigenvectors of the matrix A, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (3*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *dspev*. It also exists with a native C interface as *LAPACKE_sspev*.

4.11.68 sspevd

sspevd computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix A in packed storage. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine sspevd(JOBZ, UPLO, N, AP, W, Z, LDZ, WORK, LWORK, IWORK, LIWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void sspevd_(const char *jobz, const char *uplo, const armpl_int_t *n,
             float *ap, float *w, float *z, const armpl_int_t *ldz,
             float *work, const armpl_int_t *lwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is REAL

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n.

On exit, AP is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the diagonal and first superdiagonal of the tridiagonal matrix T overwrite the corresponding elements of A, and if UPLO = 'L', the diagonal and first subdiagonal of T overwrite the corresponding elements of A.

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is REAL

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the orthonormal eigenvectors of the matrix A, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the required LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If N <= 1, LWORK must be at least 1. If JOBZ = 'N' and N > 1, LWORK must be at least 2*N. If JOBZ = 'V' and N > 1, LWORK must be at least 1 + 6* N + N**2.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the required sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the required LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. If JOBZ = 'N' or N <= 1, LIWORK must be at least 1. If JOBZ = 'V' and N > 1, LIWORK must be at least 3 + 5*N.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the required sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *dspevd*. It also exists with a native C interface as *LAPACKE_sspevd*.

4.11.69 sspevx

sspevx computes selected eigenvalues and, optionally, eigenvectors of a real symmetric matrix A in packed storage. Eigenvalues/vectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine sspevx(JOBZ, RANGE, UPLO, N, AP, VL, VU, IL, IU, ABSTOL, M, W, Z,
                  LDZ, WORK, IWORK, IFAIL, INFO)
```

C specification:

```
#include "armpl.h"

void sspevx_(const char *jobz, const char *range, const char *uplo,
             const armpl_int_t *n, float *ap, const float *vl,
             const float *vu, const armpl_int_t *il, const armpl_int_t *iu,
             const float *abstol, armpl_int_t *m, float *w, float *z,
             const armpl_int_t *ldz, float *work, armpl_int_t *iwork,
             armpl_int_t *ifail, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found; = 'V': all eigenvalues in the half-open interval (VL,VU] will be found; = 'I': the IL-th through IU-th eigenvalues will be found.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is REAL

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n.

On exit, AP is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the diagonal and first superdiagonal of the tridiagonal matrix T overwrite the corresponding elements of A, and if UPLO = 'L', the diagonal and first subdiagonal of T overwrite the corresponding elements of A.

VL Input parameter.

VL is REAL

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is REAL

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is REAL

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing AP to tridiagonal form.

Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*SLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*SLAMCH('S').

See "Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy," by Demmel and Kahan, LAPACK Working Note #3.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the selected eigenvalues in ascending order.

Z Output parameter.

Z is REAL

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). If an eigenvector fails to converge, then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL. If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (8*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (5*N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (N)

If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, then i eigenvectors failed to converge. Their indices are stored in array IFAIL.

Related Information

For this routine in other precisions, please see *dspevx*. It also exists with a native C interface as *LAPACKE_sspevx*.

4.11.70 ssptrd

`ssptrd` reduces a real symmetric matrix A stored in packed form to symmetric tridiagonal form T by an orthogonal similarity transformation: Q^T * A * Q = T.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssptrd(UPLO, N, AP, D, E, TAU, INFO)
```

C specification:

```
#include "armpl.h"

void ssptrd_(const char *uplo, const armpl_int_t *n, float *ap, float *d,
             float *e, float *tau, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is REAL

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n. On exit, if UPLO = 'U', the diagonal and first superdiagonal of A are overwritten by the corresponding elements of the tridiagonal matrix T, and the elements above the first superdiagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors; if UPLO = 'L', the diagonal and first subdiagonal of A are over- written by the corresponding elements of the tridiagonal matrix T, and the elements below the first subdiagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors. See Further Details.

D Output parameter.

D is REAL

D is an array, dimension (N). The diagonal elements of the tridiagonal matrix T: D(i) = A(i,i).

E Output parameter.

E is REAL

E is an array, dimension (N-1). The off-diagonal elements of the tridiagonal matrix T: E(i) = A(i,i+1) if UPLO = 'U', E(i) = A(i+1,i) if UPLO = 'L'.

TAU Output parameter.

TAU is REAL

TAU is an array, dimension (N-1). The scalar factors of the elementary reflectors (see Further Details).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dsptrd*. It also exists with a native C interface as *LAPACKE_ssptrd*.

4.11.71 sstebz

sstebz computes the eigenvalues of a symmetric tridiagonal matrix T. The user may ask for all eigenvalues, all eigenvalues in the half-open interval (VL, VU], or the IL-th through IU-th eigenvalues.

To avoid overflow, the matrix must be scaled so that its largest element is no greater than overflow**(1/2) * underflow**(1/4) in absolute value, and for greatest accuracy, it should not be much smaller than that.

See W. Kahan "Accurate Eigenvalues of a Symmetric Tridiagonal Matrix", Report CS41, Computer Science Dept., Stanford University, July 21, 1966.

Syntax

Fortran specification:

```
use armpl_library

subroutine sstebz(RANGE, ORDER, N, VL, VU, IL, IU, ABSTOL, D, E, M, NSPLIT, W,
                  IBLOCK, ISPLIT, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sstebz_(const char *range, const char *order, const armpl_int_t *n,
             const float *vl, const float *vu, const armpl_int_t *il,
             const armpl_int_t *iu, const float *abstol, const float *d,
             const float *e, armpl_int_t *m, armpl_int_t *nsplit, float *w,
             armpl_int_t *iblock, armpl_int_t *isplit, float *work,
             armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': ("All") all eigenvalues will be found. = 'V': ("Value") all eigenvalues in the half-open interval (VL, VU] will be found. = 'I': ("Index") the IL-th through IU-th eigenvalues (of the entire matrix) will be found.

ORDER Input parameter.

ORDER is CHARACTER*1

= 'B': ("By Block") the eigenvalues will be grouped by split-off block (see IBLOCK, ISPLIT) and ordered from smallest to largest within the block. = 'E': ("Entire matrix") the eigenvalues for the entire matrix will be ordered from smallest to largest.

N Input parameter.

N is INTEGER

The order of the tridiagonal matrix T. N >= 0.

VL Input parameter.

VL is REAL

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. Eigenvalues less than or equal to VL, or greater than VU, will not be returned. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is REAL

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. Eigenvalues less than or equal to VL, or greater than VU, will not be returned. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is REAL

The absolute tolerance for the eigenvalues. An eigenvalue (or cluster) is considered to be located if it has been determined to lie in an interval whose width is ABSTOL or less. If ABSTOL is less than or equal to zero, then ULP*|T| will be used, where |T| means the 1-norm of T.

Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*SLAMCH('S'), not zero.

D Input parameter.

D is REAL

D is an array, dimension (N). The n diagonal elements of the tridiagonal matrix T.

E Input parameter.

E is REAL

E is an array, dimension (N-1). The (n-1) off-diagonal elements of the tridiagonal matrix T.

M Output parameter.

M is INTEGER

The actual number of eigenvalues found. 0 <= M <= N. (See also the description of INFO=2,3.)

NSPLIT Output parameter.

NSPLIT is INTEGER

The number of diagonal blocks in the matrix T. 1 <= NSPLIT <= N.

W Output parameter.

W is REAL

W is an array, dimension (N). On exit, the first M elements of W will contain the eigenvalues. (SSTEBZ may use the remaining N-M elements as workspace.)

IBLOCK Output parameter.

IBLOCK is INTEGER array, dimension (N)

At each row/column j where E(j) is zero or small, the matrix T is considered to split into a block diagonal matrix. On exit, if INFO = 0, IBLOCK(i) specifies to which block (from 1 to the number of blocks) the eigenvalue W(i) belongs. (SSTEBZ may use the remaining N-M elements as workspace.)

ISPLIT Output parameter.

ISPLIT is INTEGER array, dimension (N)

The splitting points, at which T breaks up into submatrices. The first submatrix consists of rows/columns 1 to ISPLIT(1), the second of rows/columns ISPLIT(1)+1 through ISPLIT(2), etc., and the NSPLIT-th consists of rows/columns ISPLIT(NSPLIT-1)+1 through ISPLIT(NSPLIT)=N. (Only the first NSPLIT elements will actually be used, but since the user cannot know a priori what value NSPLIT will have, N words must be reserved for ISPLIT.)

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (4*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (3*N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: some or all of the eigenvalues failed to converge or were not computed: =1 or 3: Bisection failed to converge for some eigenvalues; these eigenvalues are flagged by a negative block number. The effect is that the eigenvalues may not be as accurate as the absolute and relative tolerances. This is generally caused by unexpectedly inaccurate arithmetic. =2 or 3: RANGE='I' only: Not all of the eigenvalues IL:IU were found. Effect: M < IU+1-IL Cause: non-monotonic arithmetic, causing the Sturm sequence to be non-monotonic. Cure: recalculate, using RANGE='A', and pick out eigenvalues IL:IU. In some cases, increasing the PARAMETER "FUDGE" may make things work. = 4: RANGE='I', and the Gershgorin interval initially used was too small. No eigenvalues were computed. Probable cause: your machine has sloppy floating-point arithmetic. Cure: Increase the PARAMETER "FUDGE", recompile, and try again.

Related Information

For this routine in other precisions, please see *dstebz*. It also exists with a native C interface as *LAPACKE_sstebz*.

4.11.72 sstedc

sstedc computes all eigenvalues and, optionally, eigenvectors of a symmetric tridiagonal matrix using the divide and conquer method. The eigenvectors of a full or band real symmetric matrix can also be found if SSYTRD or SSPTRD or SSBTRD has been used to reduce this matrix to tridiagonal form.

This code makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none. See SLAED3 for details.

Syntax

Fortran specification:

```
use armpl_library

subroutine sstedc(COMPZ, N, D, E, Z, LDZ, WORK, LWORK, IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sstedc_(const char *compz, const armpl_int_t *n, float *d, float *e,
             float *z, const armpl_int_t *ldz, float *work,
             const armpl_int_t *lwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

COMPZ Input parameter.

COMPZ is CHARACTER*1

= 'N': Compute eigenvalues only. = 'I': Compute eigenvectors of tridiagonal matrix also. = 'V': Compute eigenvectors of original dense symmetric matrix also. On entry, Z contains the orthogonal matrix used to reduce the original matrix to tridiagonal form.

N Input parameter.

N is INTEGER

The dimension of the symmetric tridiagonal matrix. N >= 0.

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry, the diagonal elements of the tridiagonal matrix. On exit, if INFO = 0, the eigenvalues in ascending order.

E Input and output parameter.

E is REAL

E is an array, dimension (N-1). On entry, the subdiagonal elements of the tridiagonal matrix. On exit, E has been destroyed.

Z Input and output parameter.

Z is REAL

Z is an array, dimension (LDZ, N). On entry, if COMPZ = 'V', then Z contains the orthogonal matrix used in the reduction to tridiagonal form. On exit, if INFO = 0, then if COMPZ = 'V', Z contains the orthonormal eigenvectors of the original symmetric matrix, and if COMPZ = 'I', Z contains the orthonormal eigenvectors of the symmetric tridiagonal matrix. If COMPZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1. If eigenvectors are desired, then LDZ >= max(1, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If COMPZ = 'N' or N <= 1 then LWORK must be at least 1. If COMPZ = 'V' and N > 1 then LWORK must be at least (1 + 3* N + 2*N*lg N + 4*N**2), where lg(N) = smallest integer k such that 2**k >= N. If COMPZ = 'I' and N > 1 then LWORK must be at least (1 + 4* N + N**2). Note that for COMPZ = 'I' or 'V', then if N is less than or equal to the minimum divide size, usually 25, then LWORK need only be max(1,2*(N-1)).

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. If COMPZ = 'N' or N <= 1 then LIWORK must be at least 1. If COMPZ = 'V' and N > 1 then LIWORK must be at least (6 + 6* N + 5*N*lg N). If COMPZ = 'I' and N > 1 then LIWORK must be at least (3 + 5* N). Note that for COMPZ = 'I' or 'V', then if N is less than or equal to the minimum divide size, usually 25, then LIWORK need only be 1.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the IWORK array, returns this value as the first entry of the IWORK array, and no error message related to LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: The algorithm failed to compute an eigenvalue while working on the submatrix lying in rows and columns INFO/(N+1) through mod(INFO,N+1).

Related Information

For this routine in other precisions, please see *cstedc*, *dstedc* and *zstedc*. It also exists with a native C interface as *LAPACKE_sstedc*.

4.11.73 sstegr

sstegr computes selected eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix T. Any such unreduced matrix has a well defined set of pairwise different real eigenvalues, the corresponding real eigenvectors are pairwise orthogonal.

The spectrum may be computed either completely or partially by specifying either an interval (VL,VU] or a range of indices IL:IU for the desired eigenvalues.

sstegr is a compatibility wrapper around the improved SSTEMR routine. See SSTEMR for further details.

One important change is that the ABSTOL parameter no longer provides any benefit and hence is no longer used.

Note : sstegr and SSTEMR work only on machines which follow IEEE-754 floating-point standard in their handling of infinities and NaNs. Normal execution may create these exceptiona values and hence may abort due to a floating point exception in environments which do not conform to the IEEE-754 standard.

Syntax

Fortran specification:

```
use armpl_library

subroutine sstegr(JOBZ, RANGE, N, D, E, VL, VU, IL, IU, ABSTOL, M, W, Z, LDZ,
                  ISUPPZ, WORK, LWORK, IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sstegr_(const char *jobz, const char *range, const armpl_int_t *n,
             float *d, float *e, const float *vl, const float *vu,
             const armpl_int_t *il, const armpl_int_t *iu,
             const float *abstol, armpl_int_t *m, float *w, float *z,
             const armpl_int_t *ldz, armpl_int_t *isuppz, float *work,
             const armpl_int_t *lwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found. = 'V': all eigenvalues in the half-open interval (VL,VU] will be found. = 'I': the IL-th through IU-th eigenvalues will be found.

N Input parameter.

N is INTEGER

The order of the matrix. N >= 0.

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry, the N diagonal elements of the tridiagonal matrix T. On exit, D is overwritten.

E Input and output parameter.

E is REAL

E is an array, dimension (N). On entry, the (N-1) subdiagonal elements of the tridiagonal matrix T in elements 1 to N-1 of E. E(N) need not be set on input, but is used internally as workspace. On exit, E is overwritten.

VL Input parameter.

VL is REAL

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is REAL

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is REAL

Unused. Was the absolute error tolerance for the eigenvalues/eigenvectors in previous versions.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is REAL

W is an array, dimension (N). The first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is REAL

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', and if INFO = 0, then the first M columns of Z contain the orthonormal eigenvectors of the matrix T corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used. Supplying N columns is always safe.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', then LDZ >= max(1, N).

ISUPPZ Output parameter.

ISUPPZ is INTEGER array, dimension (2*max(1, M))

The support of the eigenvectors in Z, i.e., the indices indicating the nonzero elements in Z. The i-th computed eigenvector is nonzero only in elements ISUPPZ(2*i-1) through ISUPPZ(2*i). This is relevant in the case when the matrix is split. ISUPPZ is only accessed when JOBZ is 'V' and N > 0.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (LWORK). On exit, if INFO = 0, WORK(1) returns the optimal (and minimal) LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,18*N) if JOBZ = 'V', and LWORK >= max(1,12*N) if JOBZ = 'N'. If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (LIWORK)

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. LIWORK >= max(1,10*N) if the eigenvectors are desired, and LIWORK >= max(1,8*N) if only the eigenvalues are to be computed. If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the IWORK array, returns this value as the first entry of the IWORK array, and no error message related to LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

On exit, INFO = 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = 1X, internal error in SLARRE, if INFO = 2X, internal error in SLARRV. Here, the digit X = ABS(IINFO) < 10, where IINFO is the nonzero error code returned by SLARRE or SLARRV, respectively.

Related Information

For this routine in other precisions, please see *cstegr*, *dstegr* and *zstegr*. It also exists with a native C interface as *LAPACKE_sstegr*.

4.11.74 sstein

sstein computes the eigenvectors of a real symmetric tridiagonal matrix T corresponding to specified eigenvalues, using inverse iteration.

The maximum number of iterations allowed for each eigenvector is specified by an internal parameter MAXITS (currently set to 5).

Syntax

Fortran specification:

```
use armpl_library

subroutine sstein(N, D, E, M, W, IBLOCK, ISPLIT, Z, LDZ, WORK, IWORK, IFAIL,
                  INFO)
```

C specification:

```
#include "armpl.h"

void sstein_(const armpl_int_t *n, const float *d, const float *e,
             const armpl_int_t *m, const float *w, const armpl_int_t *iblock,
             const armpl_int_t *isplit, float *z, const armpl_int_t *ldz,
             float *work, armpl_int_t *iwork, armpl_int_t *ifail,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix. N >= 0.

D Input parameter.

D is REAL

D is an array, dimension (N). The n diagonal elements of the tridiagonal matrix T.

E Input parameter.

E is REAL

E is an array, dimension (N-1). The (n-1) subdiagonal elements of the tridiagonal matrix T, in elements 1 to N-1.

M Input parameter.

M is INTEGER

The number of eigenvectors to be found. 0 <= M <= N.

W Input parameter.

W is REAL

W is an array, dimension (N). The first M elements of W contain the eigenvalues for which eigenvectors are to be computed. The eigenvalues should be grouped by split-off block and ordered from smallest to largest within the block. (The output array W from SSTEBZ with ORDER = 'B' is expected here.)

IBLOCK Input parameter.

IBLOCK is INTEGER array, dimension (N)

The submatrix indices associated with the corresponding eigenvalues in W; IBLOCK(i)=1 if eigenvalue W(i) belongs to the first submatrix from the top, =2 if W(i) belongs to the second submatrix, etc. (The output array IBLOCK from SSTEBZ is expected here.)

ISPLIT Input parameter.

ISPLIT is INTEGER array, dimension (N)

The splitting points, at which T breaks up into submatrices. The first submatrix consists of rows/columns 1 to ISPLIT(1), the second of rows/columns ISPLIT(1)+1 through ISPLIT(2), etc. (The output array ISPLIT from SSTEBZ is expected here.)

Z Output parameter.

Z is REAL

Z is an array, dimension (LDZ, M). The computed eigenvectors. The eigenvector associated with the eigenvalue W(i) is stored in the i-th column of Z. Any vector which fails to converge is set to its current iterate after MAXITS iterations.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= max(1, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (5*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (M)

On normal exit, all elements of IFAIL are zero. If one or more eigenvectors fail to converge after MAXITS iterations, then their indices are stored in array IFAIL.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, then i eigenvectors failed to converge in MAXITS iterations. Their indices are stored in array IFAIL.

Related Information

For this routine in other precisions, please see *cstein*, *dstein* and *zstein*. It also exists with a native C interface as *LAPACKE_sstein*.

4.11.75 sstemr

sstemr computes selected eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix T. Any such unreduced matrix has a well defined set of pairwise different real eigenvalues, the corresponding real eigenvectors are pairwise orthogonal.

The spectrum may be computed either completely or partially by specifying either an interval (VL,VU] or a range of indices IL:IU for the desired eigenvalues.

Depending on the number of desired eigenvalues, these are computed either by bisection or the dqds algorithm. Numerically orthogonal eigenvectors are computed by the use of various suitable L D L^T factorizations near clusters of close eigenvalues (referred to as RRRs, Relatively Robust Representations). An informal sketch of the algorithm follows.

For each unreduced block (submatrix) of T,

```
(a) Compute T - sigma I  = L D L^T, so that L and D
    define all the wanted eigenvalues to high relative accuracy.
    This means that small relative changes in the entries of D and L
    cause only small relative changes in the eigenvalues and
    eigenvectors. The standard (unfactored) representation of the
    tridiagonal matrix T does not have this property in general.
(b) Compute the eigenvalues to suitable accuracy.
    If the eigenvectors are desired, the algorithm attains full
    accuracy of the computed eigenvalues only right before
    the corresponding vectors have to be computed, see steps c) and d).
(c) For each cluster of close eigenvalues, select a new
    shift close to the cluster, find a new factorization, and refine
    the shifted eigenvalues to suitable accuracy.
(d) For each eigenvalue with a large enough relative separation compute
    the corresponding eigenvector by forming a rank revealing twisted
    factorization. Go back to (c) for any clusters that remain.
```

For more details, see: - Inderjit S. Dhillon and Beresford N. Parlett: “Multiple representations

```
 to compute orthogonal eigenvectors of symmetric tridiagonal matrices,"
Linear Algebra and its Applications, 387(1), pp. 1-28, August 2004.
```

- Inderjit Dhillon and Beresford Parlett: “Orthogonal Eigenvectors and

```
 Relative Gaps," SIAM Journal on Matrix Analysis and Applications, Vol. 25,
2004.  Also LAPACK Working Note 154.
```

- Inderjit Dhillon: “A new O(n^2) algorithm for the symmetric

```
 tridiagonal eigenvalue/eigenvector problem",
Computer Science Division Technical Report No. UCB/CSD-97-971,
UC Berkeley, May 1997.
```

Further Details 1.‘‘sstemr‘‘ works only on machines which follow IEEE-754 floating-point standard in their handling of infinities and NaNs. This permits the use of efficient inner loops avoiding a check for zero divisors.

Syntax

Fortran specification:

```
use armpl_library

subroutine sstemr(JOBZ, RANGE, N, D, E, VL, VU, IL, IU, M, W, Z, LDZ, NZC,
                  ISUPPZ, TRYRAC, WORK, LWORK, IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sstemr_(const char *jobz, const char *range, const armpl_int_t *n,
             float *d, float *e, const float *vl, const float *vu,
             const armpl_int_t *il, const armpl_int_t *iu, armpl_int_t *m,
             float *w, float *z, const armpl_int_t *ldz,
             const armpl_int_t *nzc, armpl_int_t *isuppz, armpl_int_t *tryrac,
             float *work, const armpl_int_t *lwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= ‘N’: Compute eigenvalues only; = ‘V’: Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= ‘A’: all eigenvalues will be found. = ‘V’: all eigenvalues in the half-open interval (VL,VU] will be found. = ‘I’: the IL-th through IU-th eigenvalues will be found.

N Input parameter.

N is INTEGER

The order of the matrix. N >= 0.

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry, the N diagonal elements of the tridiagonal matrix T. On exit, D is overwritten.

E Input and output parameter.

E is REAL

E is an array, dimension (N). On entry, the (N-1) subdiagonal elements of the tridiagonal matrix T in elements 1 to N-1 of E. E(N) need not be set on input, but is used internally as workspace. On exit, E is overwritten.

VL Input parameter.

VL is REAL

If RANGE=’V’, the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = ‘A’ or ‘I’.

VU Input parameter.

VU is REAL

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0. Not referenced if RANGE = 'A' or 'V'.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is REAL

W is an array, dimension (N). The first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is REAL

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', and if INFO = 0, then the first M columns of Z contain the orthonormal eigenvectors of the matrix T corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and can be computed with a workspace query by setting NZC = -1, see below.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', then LDZ >= max(1, N).

NZC Input parameter.

NZC is INTEGER

The number of eigenvectors to be held in the array Z. If RANGE = 'A', then NZC >= max(1, N). If RANGE = 'V', then NZC >= the number of eigenvalues in (VL,VU]. If RANGE = 'I', then NZC >= IU-IL+1. If NZC = -1, then a workspace query is assumed; the routine calculates the number of columns of the array Z that are needed to hold the eigenvectors. This value is returned as the first entry of the Z array, and no error message related to NZC is issued by XERBLA.

ISUPPZ Output parameter.

ISUPPZ is INTEGER array, dimension (2*max(1, M))

The support of the eigenvectors in Z, i.e., the indices indicating the nonzero elements in Z. The i-th computed eigenvector is nonzero only in elements ISUPPZ(2*i-1) through ISUPPZ(2*i). This is relevant in the case when the matrix is split. ISUPPZ is only accessed when JOBZ is 'V' and N > 0.

TRYRAC Input and output parameter.

TRYRAC is LOGICAL

If TRYRAC.EQ..TRUE., indicates that the code should check whether the tridiagonal matrix defines its eigenvalues to high relative accuracy. If so, the code uses relative-accuracy preserving algorithms that might be (a bit) slower depending on the matrix. If the matrix does not define its eigenvalues to high relative accuracy, the code can uses possibly faster algorithms. If TRYRAC.EQ..FALSE., the code is not required to guarantee relatively accurate eigenvalues and can use the fastest possible techniques. On exit, a .TRUE. TRYRAC will be set to .FALSE. if the matrix does not define its eigenvalues to high relative accuracy.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (LWORK). On exit, if INFO = 0, WORK(1) returns the optimal (and minimal) LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,18*N) if JOBZ = 'V', and LWORK >= max(1,12*N) if JOBZ = 'N'. If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (LIWORK)

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. LIWORK >= max(1,10*N) if the eigenvectors are desired, and LIWORK >= max(1,8*N) if only the eigenvalues are to be computed. If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the IWORK array, returns this value as the first entry of the IWORK array, and no error message related to LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

On exit, INFO = 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = 1X, internal error in SLARRE, if INFO = 2X, internal error in SLARRV. Here, the digit X = ABS(IINFO) < 10, where IINFO is the nonzero error code returned by SLARRE or SLARRV, respectively.

Related Information

For this routine in other precisions, please see *cstemr*, *dstemr* and *zstemr*. It also exists with a native C interface as *LAPACKE_sstemr*.

4.11.76 ssteqr

ssteqr computes all eigenvalues and, optionally, eigenvectors of a symmetric tridiagonal matrix using the implicit QL or QR method. The eigenvectors of a full or band symmetric matrix can also be found if SSYTRD or SSPTRD or SSBTRD has been used to reduce this matrix to tridiagonal form.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssteqr(COMPZ, N, D, E, Z, LDZ, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void ssteqr_(const char *compz, const armpl_int_t *n, float *d, float *e,
             float *z, const armpl_int_t *ldz, float *work, armpl_int_t *info,
             ... );
```

Parameters

COMPZ Input parameter.

COMPZ is CHARACTER*1

= 'N': Compute eigenvalues only. = 'V': Compute eigenvalues and eigenvectors of the original symmetric matrix. On entry, Z must contain the orthogonal matrix used to reduce the original matrix to tridiagonal form. = 'I': Compute eigenvalues and eigenvectors of the tridiagonal matrix. Z is initialized to the identity matrix.

N Input parameter.

N is INTEGER

The order of the matrix. N >= 0.

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry, the diagonal elements of the tridiagonal matrix. On exit, if INFO = 0, the eigenvalues in ascending order.

E Input and output parameter.

E is REAL

E is an array, dimension (N-1). On entry, the (n-1) subdiagonal elements of the tridiagonal matrix. On exit, E has been destroyed.

Z Input and output parameter.

Z is REAL

Z is an array, dimension (LDZ, N). On entry, if COMPZ = 'V', then Z contains the orthogonal matrix used in the reduction to tridiagonal form. On exit, if INFO = 0, then if COMPZ = 'V', Z contains the orthonormal eigenvectors of the original symmetric matrix, and if COMPZ = 'I', Z contains the orthonormal eigenvectors of the symmetric tridiagonal matrix. If COMPZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if eigenvectors are desired, then LDZ >= max(1, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (max(1,2*N-2)). If COMPZ = 'N', then WORK is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: the algorithm has failed to find all the eigenvalues in a total of 30* N iterations; if INFO = i, then i elements of E have not converged to

zero; on exit, D and E contain the elements of a symmetric tridiagonal matrix which is orthogonally similar to the original matrix.

Related Information

For this routine in other precisions, please see *csteqr*, *dsteqr* and *zsteqr*. It also exists with a native C interface as *LAPACKE_ssteqr*.

4.11.77 ssterf

ssterf computes all eigenvalues of a symmetric tridiagonal matrix using the Pal-Walker-Kahan variant of the QL or QR algorithm.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssterf(N, D, E, INFO)
```

C specification:

```
#include "armpl.h"

void ssterf_(const armpl_int_t *n, float *d, float *e, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix. N >= 0.

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry, the n diagonal elements of the tridiagonal matrix. On exit, if INFO = 0, the eigenvalues in ascending order.

E Input and output parameter.

E is REAL

E is an array, dimension (N-1). On entry, the (n-1) subdiagonal elements of the tridiagonal matrix. On exit, E has been destroyed.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: the algorithm failed to find all of the eigenvalues in a total of 30* N iterations; if INFO = i, then i elements of E have not converged to zero.

Related Information

For this routine in other precisions, please see *dsterf*. It also exists with a native C interface as *LAPACKE_ssterf*.

4.11.78 sstev

`sstev` computes all eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix A.

Syntax

Fortran specification:

```
use armpl_library

subroutine sstev(JOBZ, N, D, E, Z, LDZ, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void sstev_(const char *jobz, const armpl_int_t *n, float *d, float *e,
            float *z, const armpl_int_t *ldz, float *work, armpl_int_t *info,
            ... );
```

Parameters

JOBZ Input parameter.

`JOBZ` is `CHARACTER*1`

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

N Input parameter.

`N` is `INTEGER`

The order of the matrix. `N` >= 0.

D Input and output parameter.

`D` is `REAL`

`D` is an array, dimension (`N`). On entry, the n diagonal elements of the tridiagonal matrix A. On exit, if `INFO` = 0, the eigenvalues in ascending order.

E Input and output parameter.

`E` is `REAL`

`E` is an array, dimension (N-1). On entry, the (n-1) subdiagonal elements of the tridiagonal matrix A, stored in elements 1 to N-1 of `E`. On exit, the contents of `E` are destroyed.

Z Output parameter.

`Z` is `REAL`

`Z` is an array, dimension (`LDZ`, N). If `JOBZ` = 'V', then if `INFO` = 0, `Z` contains the orthonormal eigenvectors of the matrix A, with the i-th column of Z holding the eigenvector associated with D(i). If `JOBZ` = 'N', then `Z` is not referenced.

LDZ Input parameter.

`LDZ` is `INTEGER`

The leading dimension of the array `Z`. `LDZ` >= 1, and if `JOBZ` = 'V', `LDZ` >= max(1, `N`).

WORK Output parameter.

`WORK` is `REAL`

`WORK` is an array, dimension (max(1,2*N-2)). If `JOBZ` = 'N', `WORK` is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of E did not converge to zero.

Related Information

For this routine in other precisions, please see *dstev*. It also exists with a native C interface as *LAPACKE_sstev*.

4.11.79 sstevd

sstevd computes all eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine sstevd(JOBZ, N, D, E, Z, LDZ, WORK, LWORK, IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sstevd_(const char *jobz, const armpl_int_t *n, float *d, float *e,
             float *z, const armpl_int_t *ldz, float *work,
             const armpl_int_t *lwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

N Input parameter.

N is INTEGER

The order of the matrix. N >= 0.

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry, the n diagonal elements of the tridiagonal matrix A. On exit, if INFO = 0, the eigenvalues in ascending order.

E Input and output parameter.

E is REAL

E is an array, dimension (N-1). On entry, the (n-1) subdiagonal elements of the tridiagonal matrix A, stored in elements 1 to N-1 of E. On exit, the contents of E are destroyed.

Z Output parameter.

Z is REAL

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the orthonormal eigenvectors of the matrix A, with the i-th column of Z holding the eigenvector associated with D(i). If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is REAL

WORK is an array,. dimension (LWORK) On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If JOBZ = 'N' or N <= 1 then LWORK must be at least 1. If JOBZ = 'V' and N > 1 then LWORK must be at least (1 + 4* N + N**2).

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. If JOBZ = 'N' or N <= 1 then LIWORK must be at least 1. If JOBZ = 'V' and N > 1 then LIWORK must be at least 3+5*N.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of E did not converge to zero.

Related Information

For this routine in other precisions, please see *dstevd*. It also exists with a native C interface as *LAPACKE_sstevd*.

4.11.80 sstevr

sstevr computes selected eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix T. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Whenever possible, sstevr calls SSTEMR to compute the eigenspectrum using Relatively Robust Representations. SSTEMR computes eigenvalues by the dqds algorithm, while orthogonal eigenvectors are computed from various "good" L D L^T representations (also known as Relatively Robust Representations). Gram-Schmidt orthogonalization is avoided as far as possible. More specifically, the various steps of the algorithm are as follows. For the i-th unreduced block of T,

```
(a) Compute T - sigma_i = L_i D_i L_i^T, such that L_i D_i L_i^T
     is a relatively robust representation,
(b) Compute the eigenvalues, lambda_j, of L_i D_i L_i^T to high
    relative accuracy by the dqds algorithm,
(c) If there is a cluster of close eigenvalues, "choose" sigma_i
    close to the cluster, and go to step (a),
(d) Given the approximate eigenvalue lambda_j of L_i D_i L_i^T,
    compute the corresponding eigenvector by forming a
    rank-revealing twisted factorization.
```

The desired accuracy of the output can be specified by the input parameter ABSTOL.

For more details, see "A new O(n^2) algorithm for the symmetric tridiagonal eigenvalue/eigenvector problem", by Inderjit Dhillon, Computer Science Division Technical Report No. UCB//CSD-97-971, UC Berkeley, May 1997.

Note 1 : sstevr calls SSTEMR when the full spectrum is requested on machines which conform to the ieee-754 floating point standard. sstevr calls SSTEBZ and SSTEIN on non-ieee machines and when partial spectrum requests are made.

Normal execution of SSTEMR may create NaNs and infinities and hence may abort due to a floating point exception in environments which do not handle NaNs and infinities in the ieee standard default manner.

Syntax

Fortran specification:

```
use armpl_library

subroutine sstevr(JOBZ, RANGE, N, D, E, VL, VU, IL, IU, ABSTOL, M, W, Z, LDZ,
                  ISUPPZ, WORK, LWORK, IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sstevr_(const char *jobz, const char *range, const armpl_int_t *n,
             float *d, float *e, const float *vl, const float *vu,
             const armpl_int_t *il, const armpl_int_t *iu,
             const float *abstol, armpl_int_t *m, float *w, float *z,
             const armpl_int_t *ldz, armpl_int_t *isuppz, float *work,
             const armpl_int_t *lwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found. = 'V': all eigenvalues in the half-open interval (VL,VU] will be found. = 'I': the IL-th through IU-th eigenvalues will be found. For RANGE = 'V' or 'I' and IU - IL < N - 1, SSTEBZ and SSTEIN are called

N Input parameter.

N is INTEGER

The order of the matrix. N >= 0.

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry, the n diagonal elements of the tridiagonal matrix A. On exit, D may be multiplied by a constant factor chosen to avoid over/underflow in computing the eigenvalues.

E Input and output parameter.

E is REAL

E is an array, dimension (max(1,N-1)). On entry, the (n-1) subdiagonal elements of the tridiagonal matrix A in elements 1 to N-1 of E. On exit, E may be multiplied by a constant factor chosen to avoid over/underflow in computing the eigenvalues.

VL Input parameter.

VL is REAL

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is REAL

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is REAL

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing A to tridiagonal form.

See "Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy," by Demmel and Kahan, LAPACK Working Note #3.

If high relative accuracy is important, set ABSTOL to SLAMCH('Safe minimum'). Doing so will guarantee that eigenvalues are computed to high relative accuracy when possible in future releases. The current code does not make any guarantees about high relative accuracy, but future releases will. See J. Barlow and J. Demmel, "Computing Accurate Eigensystems of Scaled Diagonally Dominant Matrices", LAPACK Working Note #7, for a discussion of which matrices define their eigenvalues to high relative accuracy.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is REAL

W is an array, dimension (N). The first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is REAL

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

ISUPPZ Output parameter.

ISUPPZ is INTEGER array, dimension (2*max(1, M))

The support of the eigenvectors in Z, i.e., the indices indicating the nonzero elements in Z. The i-th eigenvector is nonzero only in elements ISUPPZ(2*i-1) through ISUPPZ(2*i). Implemented only for RANGE = 'A' or 'I' and IU - IL = N - 1

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal (and minimal) LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= 20*N.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal (and minimal) LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. LIWORK >= 10*N.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: Internal error

Related Information

For this routine in other precisions, please see *dstevr*. It also exists with a native C interface as *LAPACKE_sstevr*.

4.11.81 sstevx

sstevx computes selected eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix A. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine sstevx(JOBZ, RANGE, N, D, E, VL, VU, IL, IU, ABSTOL, M, W, Z, LDZ,
                  WORK, IWORK, IFAIL, INFO)
```

C specification:

```
#include "armpl.h"

void sstevx_(const char *jobz, const char *range, const armpl_int_t *n,
             float *d, float *e, const float *vl, const float *vu,
             const armpl_int_t *il, const armpl_int_t *iu,
             const float *abstol, armpl_int_t *m, float *w, float *z,
             const armpl_int_t *ldz, float *work, armpl_int_t *iwork,
             armpl_int_t *ifail, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found. = 'V': all eigenvalues in the half-open interval (VL,VU] will be found. = 'I': the IL-th through IU-th eigenvalues will be found.

N Input parameter.

N is INTEGER

The order of the matrix. N >= 0.

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry, the n diagonal elements of the tridiagonal matrix A. On exit, D may be multiplied by a constant factor chosen to avoid over/underflow in computing the eigenvalues.

E Input and output parameter.

E is REAL

E is an array, dimension (max(1,N-1)). On entry, the (n-1) subdiagonal elements of the tridiagonal matrix A in elements 1 to N-1 of E. On exit, E may be multiplied by a constant factor chosen to avoid over/underflow in computing the eigenvalues.

VL Input parameter.

VL is REAL

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is REAL

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is REAL

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix.

Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*SLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*SLAMCH('S').

See "Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy," by Demmel and Kahan, LAPACK Working Note #3.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is REAL

W is an array, dimension (N). The first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is REAL

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). If an eigenvector fails to converge (INFO > 0), then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL. If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (5*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (5*N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (N)

If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, then i eigenvectors failed to converge. Their indices are stored in array IFAIL.

Related Information

For this routine in other precisions, please see *dstevx*. It also exists with a native C interface as *LAPACKE_sstevx*.

4.11.82 ssyev

ssyev computes all eigenvalues and, optionally, eigenvectors of a real symmetric matrix A.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssyev(JOBZ, UPLO, N, A, LDA, W, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ssyev_(const char *jobz, const char *uplo, const armpl_int_t *n,
            float *a, const armpl_int_t *lda, float *w, float *work,
            const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A. On exit, if JOBZ = 'V', then if INFO = 0, A contains the orthonormal eigenvectors of the matrix A. If JOBZ = 'N', then on exit the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= max(1,3*N-1). For optimal efficiency, LWORK >= (NB+2)*N, where NB is the blocksize for SSYTRD returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *dsyev*. It also exists with a native C interface as *LAPACKE_ssyev*.

4.11.83 ssyevd

ssyevd computes all eigenvalues and, optionally, eigenvectors of a real symmetric matrix A. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Because of large use of BLAS of level 3, ssyevd needs N**2 more workspace than SSYEVX.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssyevd(JOBZ, UPLO, N, A, LDA, W, WORK, LWORK, IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ssyevd_(const char *jobz, const char *uplo, const armpl_int_t *n,
             float *a, const armpl_int_t *lda, float *w, float *work,
             const armpl_int_t *lwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading

N-by-N lower triangular part of A contains the lower triangular part of the matrix A. On exit, if JOBZ = 'V', then if INFO = 0, A contains the orthonormal eigenvectors of the matrix A. If JOBZ = 'N', then on exit the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

WORK Output parameter.

WORK is REAL

WORK is an array,. dimension (LWORK) On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If N <= 1, LWORK must be at least 1. If JOBZ = 'N' and N > 1, LWORK must be at least 2*N+1. If JOBZ = 'V' and N > 1, LWORK must be at least 1 + 6* N + 2*N**2.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. If N <= 1, LIWORK must be at least 1. If JOBZ = 'N' and N > 1, LIWORK must be at least 1. If JOBZ = 'V' and N > 1, LIWORK must be at least 3 + 5*N.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i and JOBZ = 'N', then the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero; if INFO = i and JOBZ = 'V', then the algorithm failed to compute an eigenvalue while working on the submatrix lying in rows and columns INFO/(N+1) through mod(INFO,N+1).

Related Information

For this routine in other precisions, please see *dsyevd*. It also exists with a native C interface as *LAPACKE_ssyevd*.

4.11.84 ssyevr

ssyevr computes selected eigenvalues and, optionally, eigenvectors of a real symmetric matrix A. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

ssyevr first reduces the matrix A to tridiagonal form T with a call to SSYTRD. Then, whenever possible, ssyevr calls SSTEMR to compute the eigenspectrum using Relatively Robust Representations. SSTEMR computes eigenvalues by the dqds algorithm, while orthogonal eigenvectors are computed from various "good" L D L^T representations (also known as Relatively Robust Representations). Gram-Schmidt orthogonalization is avoided as far as possible. More specifically, the various steps of the algorithm are as follows.

For each unreduced block (submatrix) of T,

```
(a) Compute T - sigma I  = L D L^T, so that L and D
    define all the wanted eigenvalues to high relative accuracy.
    This means that small relative changes in the entries of D and L
    cause only small relative changes in the eigenvalues and
    eigenvectors. The standard (unfactored) representation of the
    tridiagonal matrix T does not have this property in general.
(b) Compute the eigenvalues to suitable accuracy.
    If the eigenvectors are desired, the algorithm attains full
    accuracy of the computed eigenvalues only right before
    the corresponding vectors have to be computed, see steps c) and d).
(c) For each cluster of close eigenvalues, select a new
    shift close to the cluster, find a new factorization, and refine
    the shifted eigenvalues to suitable accuracy.
(d) For each eigenvalue with a large enough relative separation compute
    the corresponding eigenvector by forming a rank revealing twisted
    factorization. Go back to (c) for any clusters that remain.
```

The desired accuracy of the output can be specified by the input parameter ABSTOL.

For more details, see SSTEMR's documentation and: - Inderjit S. Dhillon and Beresford N. Parlett: "Multiple representations

```
 to compute orthogonal eigenvectors of symmetric tridiagonal matrices,"
Linear Algebra and its Applications, 387(1), pp. 1-28, August 2004.
```

- Inderjit Dhillon and Beresford Parlett: "Orthogonal Eigenvectors and

```
 Relative Gaps," SIAM Journal on Matrix Analysis and Applications, Vol. 25,
2004.  Also LAPACK Working Note 154.
```

- Inderjit Dhillon: "A new O(n^2) algorithm for the symmetric

```
 tridiagonal eigenvalue/eigenvector problem",
Computer Science Division Technical Report No. UCB/CSD-97-971,
UC Berkeley, May 1997.
```

Note 1 : ssyevr calls SSTEMR when the full spectrum is requested on machines which conform to the ieee-754 floating point standard. ssyevr calls SSTEBZ and SSTEIN on non-ieee machines and when partial spectrum requests are made.

Normal execution of SSTEMR may create NaNs and infinities and hence may abort due to a floating point exception in environments which do not handle NaNs and infinities in the ieee standard default manner.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssyevr(JOBZ, RANGE, UPLO, N, A, LDA, VL, VU, IL, IU, ABSTOL, M, W,
                  Z, LDZ, ISUPPZ, WORK, LWORK, IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ssyevr_(const char *jobz, const char *range, const char *uplo,
             const armpl_int_t *n, float *a, const armpl_int_t *lda,
             const float *vl, const float *vu, const armpl_int_t *il,
             const armpl_int_t *iu, const float *abstol, armpl_int_t *m,
             float *w, float *z, const armpl_int_t *ldz, armpl_int_t *isuppz,
             float *work, const armpl_int_t *lwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found. = 'V': all eigenvalues in the half-open interval (VL,VU] will be found. = 'I': the IL-th through IU-th eigenvalues will be found. For RANGE = 'V' or 'I' and IU - IL < N - 1, SSTEBZ and SSTEIN are called

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A. On exit, the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

VL Input parameter.

VL is REAL

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is REAL

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is REAL

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing A to tridiagonal form.

See "Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy," by Demmel and Kahan, LAPACK Working Note #3.

If high relative accuracy is important, set ABSTOL to SLAMCH('Safe minimum'). Doing so will guarantee that eigenvalues are computed to high relative accuracy when possible in future releases. The current code does not make any guarantees about high relative accuracy, but future releases will. See J. Barlow and J. Demmel, "Computing Accurate Eigensystems of Scaled Diagonally Dominant Matrices", LAPACK Working Note #7, for a discussion of which matrices define their eigenvalues to high relative accuracy.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is REAL

W is an array, dimension (N). The first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is REAL

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used. Supplying N columns is always safe.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

ISUPPZ Output parameter.

ISUPPZ is INTEGER array, dimension (2*max(1, M))

The support of the eigenvectors in Z, i.e., the indices indicating the nonzero elements in Z. The i-th eigenvector is nonzero only in elements ISUPPZ(2*i-1) through ISUPPZ(2*i). This is an output of SSTEMR (tridiagonal matrix). The support of the eigenvectors of A is typically 1:N because of the orthogonal transformations applied by SORMTR. Implemented only for RANGE = 'A' or 'I' and IU - IL = N - 1

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,26*N). For optimal efficiency, LWORK >= (NB+6)*N, where NB is the max of the blocksize for SSYTRD and SORMTR returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. LIWORK >= max(1,10*N).

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: Internal error

Related Information

For this routine in other precisions, please see *dsyevr*. It also exists with a native C interface as *LAPACKE_ssyevr*.

4.11.85 ssyevx

ssyevx computes selected eigenvalues and, optionally, eigenvectors of a real symmetric matrix A. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssyevx(JOBZ, RANGE, UPLO, N, A, LDA, VL, VU, IL, IU, ABSTOL, M, W,
                  Z, LDZ, WORK, LWORK, IWORK, IFAIL, INFO)
```

C specification:

```
#include "armpl.h"

void ssyevx_(const char *jobz, const char *range, const char *uplo,
             const armpl_int_t *n, float *a, const armpl_int_t *lda,
```

(continues on next page)

(continued from previous page)

```
                const float *vl, const float *vu, const armpl_int_t *il,
                const armpl_int_t *iu, const float *abstol, armpl_int_t *m,
                float *w, float *z, const armpl_int_t *ldz, float *work,
                const armpl_int_t *lwork, armpl_int_t *iwork, armpl_int_t *ifail,
                armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found. = 'V': all eigenvalues in the half-open interval (VL,VU] will be found. = 'I': the IL-th through IU-th eigenvalues will be found.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A. On exit, the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

VL Input parameter.

VL is REAL

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is REAL

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is REAL

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing A to tridiagonal form.

Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*SLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*SLAMCH('S').

See "Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy," by Demmel and Kahan, LAPACK Working Note #3.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is REAL

W is an array, dimension (N). On normal exit, the first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is REAL

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). If an eigenvector fails to converge, then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL. If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= 1, when N <= 1; otherwise 8*N. For optimal efficiency, LWORK >= (NB+3)*N, where NB is the max of the blocksize for SSYTRD and SORMTR returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (5*N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (N)

If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, then i eigenvectors failed to converge. Their indices are stored in array IFAIL.

Related Information

For this routine in other precisions, please see *dsyevx*. It also exists with a native C interface as *LAPACKE_ssyevx*.

4.11.86 ssytrd

ssytrd reduces a real symmetric matrix A to real symmetric tridiagonal form T by an orthogonal similarity transformation: $Q^T * A * Q = T$.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssytrd(UPLO, N, A, LDA, D, E, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ssytrd_(const char *uplo, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, float *d, float *e, float *tau,
             float *work, const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, if UPLO = 'U', the diagonal and first superdiagonal of A are overwritten by the corresponding elements of the tridiagonal matrix T, and the elements above the first superdiagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors; if UPLO = 'L', the diagonal and first subdiagonal of A are over- written by the corresponding elements of the tridiagonal matrix T, and the elements below the first subdiagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

D Output parameter.

D is REAL

D is an array, dimension (N). The diagonal elements of the tridiagonal matrix T: D(i) = A(i,i).

E Output parameter.

E is REAL

E is an array, dimension (N-1). The off-diagonal elements of the tridiagonal matrix T: E(i) = A(i,i+1) if UPLO = 'U', E(i) = A(i+1,i) if UPLO = 'L'.

TAU Output parameter.

TAU is REAL

TAU is an array, dimension (N-1). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= 1. For optimum performance LWORK >= N*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dsytrd*. It also exists with a native C interface as *LAPACKE_ssytrd*.

4.11.87 zhbev

zhbev computes all the eigenvalues and, optionally, eigenvectors of a complex Hermitian band matrix A.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhbev(JOBZ, UPLO, N, KD, AB, LDAB, W, Z, LDZ, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhbev_(const char *jobz, const char *uplo, const armpl_int_t *n,
            const armpl_int_t *kd, armpl_doublecomplex_t *ab,
            const armpl_int_t *ldab, double *w, armpl_doublecomplex_t *z,
            const armpl_int_t *ldz, armpl_doublecomplex_t *work,
            double *rwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the Hermitian band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, AB is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the first superdiagonal and the diagonal of the tridiagonal matrix T are returned in rows KD and KD+1 of AB, and if UPLO = 'L', the diagonal and first subdiagonal of T are returned in the first two rows of AB.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD + 1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the orthonormal eigenvectors of the matrix A, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (max(1,3*N-2)) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *chbev*. It also exists with a native C interface as *LAPACKE_zhbev*.

4.11.88 zhbevd

zhbevd computes all the eigenvalues and, optionally, eigenvectors of a complex Hermitian band matrix A. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhbevd(JOBZ, UPLO, N, KD, AB, LDAB, W, Z, LDZ, WORK, LWORK, RWORK,
                  LRWORK, IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhbevd_(const char *jobz, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *kd, armpl_doublecomplex_t *ab,
             const armpl_int_t *ldab, double *w, armpl_doublecomplex_t *z,
             const armpl_int_t *ldz, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, double *rwork,
             const armpl_int_t *lrwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the Hermitian band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, AB is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the first superdiagonal and the diagonal of the tridiagonal matrix T are returned in rows KD and KD+1 of AB, and if UPLO = 'L', the diagonal and first subdiagonal of T are returned in the first two rows of AB.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD + 1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the orthonormal eigenvectors of the matrix A, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If N <= 1, LWORK must be at least 1. If JOBZ = 'N' and N > 1, LWORK must be at least N. If JOBZ = 'V' and N > 1, LWORK must be at least 2*N**2.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array,. dimension (LRWORK) On exit, if INFO = 0, RWORK(1) returns the optimal LRWORK.

LRWORK Input parameter.

LRWORK is INTEGER

The dimension of array RWORK. If N <= 1, LRWORK must be at least 1. If JOBZ = 'N' and N > 1, LRWORK must be at least N. If JOBZ = 'V' and N > 1, LRWORK must be at least 1 + 5* N + 2*N**2.

If LRWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of array IWORK. If JOBZ = 'N' or N <= 1, LIWORK must be at least 1. If JOBZ = 'V' and N > 1, LIWORK must be at least 3 + 5* N .

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *chbevd*. It also exists with a native C interface as *LAPACKE_zhbevd*.

4.11.89 zhbevx

zhbevx computes selected eigenvalues and, optionally, eigenvectors of a complex Hermitian band matrix A. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhbevx(JOBZ, RANGE, UPLO, N, KD, AB, LDAB, Q, LDQ, VL, VU, IL, IU,
                  ABSTOL, M, W, Z, LDZ, WORK, RWORK, IWORK, IFAIL, INFO)
```

C specification:

```
#include "armpl.h"

void zhbevx_(const char *jobz, const char *range, const char *uplo,
             const armpl_int_t *n, const armpl_int_t *kd,
             armpl_doublecomplex_t *ab, const armpl_int_t *ldab,
             armpl_doublecomplex_t *q, const armpl_int_t *ldq,
             const double *vl, const double *vu, const armpl_int_t *il,
             const armpl_int_t *iu, const double *abstol, armpl_int_t *m,
             double *w, armpl_doublecomplex_t *z, const armpl_int_t *ldz,
             armpl_doublecomplex_t *work, double *rwork, armpl_int_t *iwork,
             armpl_int_t *ifail, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found; = 'V': all eigenvalues in the half-open interval (VL,VU] will be found; = 'I': the IL-th through IU-th eigenvalues will be found.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the Hermitian band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, AB is overwritten by values generated during the reduction to tridiagonal form.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD + 1.

Q Output parameter.

Q is COMPLEX*16

Q is an array, dimension (LDQ, N). If JOBZ = 'V', the N-by-N unitary matrix used in the reduction to tridiagonal form. If JOBZ = 'N', the array Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. If JOBZ = 'V', then LDQ >= max(1, N).

VL Input parameter.

VL is DOUBLE PRECISION

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is DOUBLE PRECISION

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is DOUBLE PRECISION

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing AB to tridiagonal form.

Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*DLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*DLAMCH('S').

See "Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy," by Demmel and Kahan, LAPACK Working Note #3.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). The first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). If an eigenvector fails to converge, then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL. If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (7*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (5*N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (N)

If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, then i eigenvectors failed to converge. Their indices are stored in array IFAIL.

Related Information

For this routine in other precisions, please see *chbevx*. It also exists with a native C interface as *LAPACKE_zhbevx*.

4.11.90 zhbtrd

zhbtrd reduces a complex Hermitian band matrix A to real symmetric tridiagonal form T by a unitary similarity transformation: $Q^H * A * Q = T$.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhbtrd(VECT, UPLO, N, KD, AB, LDAB, D, E, Q, LDQ, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhbtrd_(const char *vect, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *kd, armpl_doublecomplex_t *ab,
             const armpl_int_t *ldab, double *d, double *e,
             armpl_doublecomplex_t *q, const armpl_int_t *ldq,
             armpl_doublecomplex_t *work, armpl_int_t *info, ... );
```

Parameters

VECT Input parameter.

VECT is CHARACTER*1

= 'N': do not form Q; = 'V': form Q; = 'U': update a matrix X, by forming X*Q.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the Hermitian band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd). On exit, the diagonal elements of AB are overwritten by the diagonal elements of the tridiagonal matrix T; if KD > 0, the elements on the first superdiagonal (if UPLO = 'U') or the first subdiagonal (if UPLO = 'L') are overwritten by the off-diagonal elements of T; the rest of AB is overwritten by values generated during the reduction.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

D Output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The diagonal elements of the tridiagonal matrix T.

E Output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N-1). The off-diagonal elements of the tridiagonal matrix T: E(i) = T(i,i+1) if UPLO = 'U'; E(i) = T(i+1,i) if UPLO = 'L'.

Q Input and output parameter.

Q is COMPLEX*16

Q is an array, dimension (LDQ, N). On entry, if VECT = 'U', then Q must contain an N-by-N matrix X; if VECT = 'N' or 'V', then Q need not be set.

On exit: if VECT = 'V', Q contains the N-by-N unitary matrix Q; if VECT = 'U', Q contains the product X*Q; if VECT = 'N', the array Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= 1, and LDQ >= N if VECT = 'V' or 'U'.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *chbtrd*. It also exists with a native C interface as *LAPACKE_zhbtrd*.

4.11.91 zheev

zheev computes all eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix A.

Syntax

Fortran specification:

```
use armpl_library

subroutine zheev(JOBZ, UPLO, N, A, LDA, W, WORK, LWORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zheev_(const char *jobz, const char *uplo, const armpl_int_t *n,
            armpl_doublecomplex_t *a, const armpl_int_t *lda, double *w,
```

(continues on next page)

(continued from previous page)

```
            armpl_doublecomplex_t *work, const armpl_int_t *lwork,
            double *rwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A. On exit, if JOBZ = 'V', then if INFO = 0, A contains the orthonormal eigenvectors of the matrix A. If JOBZ = 'N', then on exit the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= max(1,2*N-1). For optimal efficiency, LWORK >= (NB+1)*N, where NB is the blocksize for ZHETRD returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (max(1, 3*N-2)) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *cheev*. It also exists with a native C interface as *LAPACKE_zheev*.

4.11.92 zheevd

zheevd computes all eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix A. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine zheevd(JOBZ, UPLO, N, A, LDA, W, WORK, LWORK, RWORK, LRWORK, IWORK,
                  LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zheevd_(const char *jobz, const char *uplo, const armpl_int_t *n,
             armpl_doublecomplex_t *a, const armpl_int_t *lda, double *w,
             armpl_doublecomplex_t *work, const armpl_int_t *lwork,
             double *rwork, const armpl_int_t *lrwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A. On exit, if JOBZ = 'V', then if INFO = 0, A contains the orthonormal eigenvectors of the matrix A. If JOBZ = 'N', then on exit the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. If N <= 1, LWORK must be at least 1. If JOBZ = 'N' and N > 1, LWORK must be at least N + 1. If JOBZ = 'V' and N > 1, LWORK must be at least 2* N + N**2.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array,. dimension (LRWORK) On exit, if INFO = 0, RWORK(1) returns the optimal LRWORK.

LRWORK Input parameter.

LRWORK is INTEGER

The dimension of the array RWORK. If N <= 1, LRWORK must be at least 1. If JOBZ = 'N' and N > 1, LRWORK must be at least N. If JOBZ = 'V' and N > 1, LRWORK must be at least 1 + 5* N + 2*N**2.

If LRWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. If N <= 1, LIWORK must be at least 1. If JOBZ = 'N' and N > 1, LIWORK must be at least 1. If JOBZ = 'V' and N > 1, LIWORK must be at least 3 + 5*N.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i and JOBZ = 'N', then the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero; if INFO = i and JOBZ = 'V', then the algorithm failed to compute an eigenvalue while working on the submatrix lying in rows and columns INFO/(N+1) through mod(INFO,N+1).

Related Information

For this routine in other precisions, please see *cheevd*. It also exists with a native C interface as *LAPACKE_zheevd*.

4.11.93 zheevr

zheevr computes selected eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix A. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

zheevr first reduces the matrix A to tridiagonal form T with a call to ZHETRD. Then, whenever possible, zheevr calls ZSTEMR to compute eigenspectrum using Relatively Robust Representations. ZSTEMR computes eigenvalues by the dqds algorithm, while orthogonal eigenvectors are computed from various "good" L D L^T representations (also known as Relatively Robust Representations). Gram-Schmidt orthogonalization is avoided as far as possible. More specifically, the various steps of the algorithm are as follows.

For each unreduced block (submatrix) of T,

```
(a) Compute T - sigma I  = L D L^T, so that L and D
    define all the wanted eigenvalues to high relative accuracy.
    This means that small relative changes in the entries of D and L
    cause only small relative changes in the eigenvalues and
    eigenvectors. The standard (unfactored) representation of the
    tridiagonal matrix T does not have this property in general.
(b) Compute the eigenvalues to suitable accuracy.
    If the eigenvectors are desired, the algorithm attains full
    accuracy of the computed eigenvalues only right before
    the corresponding vectors have to be computed, see steps c) and d).
(c) For each cluster of close eigenvalues, select a new
    shift close to the cluster, find a new factorization, and refine
    the shifted eigenvalues to suitable accuracy.
(d) For each eigenvalue with a large enough relative separation compute
    the corresponding eigenvector by forming a rank revealing twisted
    factorization. Go back to (c) for any clusters that remain.
```

The desired accuracy of the output can be specified by the input parameter ABSTOL.

For more details, see DSTEMR's documentation and: - Inderjit S. Dhillon and Beresford N. Parlett: "Multiple representations

```
 to compute orthogonal eigenvectors of symmetric tridiagonal matrices,"
Linear Algebra and its Applications, 387(1), pp. 1-28, August 2004.
```

- Inderjit Dhillon and Beresford Parlett: "Orthogonal Eigenvectors and

```
 Relative Gaps," SIAM Journal on Matrix Analysis and Applications, Vol. 25,
2004.  Also LAPACK Working Note 154.
```

- Inderjit Dhillon: "A new O(n^2) algorithm for the symmetric

```
 tridiagonal eigenvalue/eigenvector problem",
Computer Science Division Technical Report No. UCB/CSD-97-971,
UC Berkeley, May 1997.
```

Note 1 : zheevr calls ZSTEMR when the full spectrum is requested on machines which conform to the ieee-754 floating point standard. zheevr calls DSTEBZ and ZSTEIN on non-ieee machines and when partial spectrum requests are made.

Normal execution of ZSTEMR may create NaNs and infinities and hence may abort due to a floating point exception in environments which do not handle NaNs and infinities in the ieee standard default manner.

Syntax

Fortran specification:

```
use armpl_library

subroutine zheevr(JOBZ, RANGE, UPLO, N, A, LDA, VL, VU, IL, IU, ABSTOL, M, W,
                  Z, LDZ, ISUPPZ, WORK, LWORK, RWORK, LRWORK, IWORK, LIWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void zheevr_(const char *jobz, const char *range, const char *uplo,
             const armpl_int_t *n, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, const double *vl, const double *vu,
             const armpl_int_t *il, const armpl_int_t *iu,
             const double *abstol, armpl_int_t *m, double *w,
             armpl_doublecomplex_t *z, const armpl_int_t *ldz,
             armpl_int_t *isuppz, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, double *rwork,
             const armpl_int_t *lrwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found. = 'V': all eigenvalues in the half-open interval (VL,VU] will be found. = 'I': the IL-th through IU-th eigenvalues will be found. For RANGE = 'V' or 'I' and IU - IL < N - 1, DSTEBZ and ZSTEIN are called

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A. On exit, the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

VL Input parameter.

VL is DOUBLE PRECISION

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is DOUBLE PRECISION

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is DOUBLE PRECISION

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing A to tridiagonal form.

See "Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy," by Demmel and Kahan, LAPACK Working Note #3.

If high relative accuracy is important, set ABSTOL to DLAMCH('Safe minimum'). Doing so will guarantee that eigenvalues are computed to high relative accuracy when possible in future releases. The current code does not make any guarantees about high relative accuracy, but furutre releases will. See J. Barlow and J. Demmel, "Computing Accurate Eigensystems of Scaled Diagonally Dominant Matrices", LAPACK Working Note #7, for a discussion of which matrices define their eigenvalues to high relative accuracy.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). The first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

ISUPPZ Output parameter.

ISUPPZ is INTEGER array, dimension (2*max(1, M))

The support of the eigenvectors in Z, i.e., the indices indicating the nonzero elements in Z. The i-th eigenvector is nonzero only in elements ISUPPZ(2*i-1) through ISUPPZ(2*i). This is an output of ZSTEMR (tridiagonal matrix). The support of the eigenvectors of A is typically 1:N because of the unitary transformations applied by ZUNMTR. Implemented only for RANGE = 'A' or 'I' and IU - IL = N - 1

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= max(1,2*N). For optimal efficiency, LWORK >= (NB+1)*N, where NB is the max of the blocksize for ZHETRD and for ZUNMTR as returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (MAX(1, LRWORK)). On exit, if INFO = 0, RWORK(1) returns the optimal (and minimal) LRWORK.

LRWORK Input parameter.

LRWORK is INTEGER

The length of the array RWORK. LRWORK >= max(1,24*N).

If LRWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal (and minimal) LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. LIWORK >= max(1,10*N).

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: Internal error

Related Information

For this routine in other precisions, please see *cheevr*. It also exists with a native C interface as *LAPACKE_zheevr*.

4.11.94 zheevx

zheevx computes selected eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix A. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine zheevx(JOBZ, RANGE, UPLO, N, A, LDA, VL, VU, IL, IU, ABSTOL, M, W,
                  Z, LDZ, WORK, LWORK, RWORK, IWORK, IFAIL, INFO)
```

C specification:

```
#include "armpl.h"

void zheevx_(const char *jobz, const char *range, const char *uplo,
             const armpl_int_t *n, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, const double *vl, const double *vu,
             const armpl_int_t *il, const armpl_int_t *iu,
             const double *abstol, armpl_int_t *m, double *w,
             armpl_doublecomplex_t *z, const armpl_int_t *ldz,
             armpl_doublecomplex_t *work, const armpl_int_t *lwork,
             double *rwork, armpl_int_t *iwork, armpl_int_t *ifail,
             armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= ‘A’: all eigenvalues will be found. = ‘V’: all eigenvalues in the half-open interval (VL,VU] will be found. = ‘I’: the IL-th through IU-th eigenvalues will be found.

UPLO Input parameter.

UPLO is CHARACTER*1

= ‘U’: Upper triangle of A is stored; = ‘L’: Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = ‘U’, the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = ‘L’, the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A. On exit, the lower triangle (if UPLO=’L’) or the upper triangle (if UPLO=’U’) of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

VL Input parameter.

VL is DOUBLE PRECISION

If RANGE=’V’, the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = ‘A’ or ‘I’.

VU Input parameter.

VU is DOUBLE PRECISION

If RANGE=’V’, the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = ‘A’ or ‘I’.

IL Input parameter.

IL is INTEGER

If RANGE=’I’, the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = ‘A’ or ‘V’.

IU Input parameter.

IU is INTEGER

If RANGE=’I’, the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = ‘A’ or ‘V’.

ABSTOL Input parameter.

ABSTOL is DOUBLE PRECISION

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing A to tridiagonal form.

Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*DLAMCH(‘S’), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*DLAMCH(‘S’).

See "Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy," by Demmel and Kahan, LAPACK Working Note #3.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). On normal exit, the first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). If an eigenvector fails to converge, then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL. If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= 1, when N <= 1; otherwise 2*N. For optimal efficiency, LWORK >= (NB+1)*N, where NB is the max of the blocksize for ZHETRD and for ZUNMTR as returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (7*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (5*N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (N)

If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, then i eigenvectors failed to converge. Their indices are stored in array IFAIL.

Related Information

For this routine in other precisions, please see *cheevx*. It also exists with a native C interface as *LAPACKE_zheevx*.

4.11.95 zhetrd

zhetrd reduces a complex Hermitian matrix A to real symmetric tridiagonal form T by a unitary similarity transformation: $Q^H * A * Q = T$.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhetrd(UPLO, N, A, LDA, D, E, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhetrd_(const char *uplo, const armpl_int_t *n, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, double *d, double *e,
             armpl_doublecomplex_t *tau, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, if UPLO = 'U', the diagonal and first superdiagonal of A are overwritten by the corresponding elements of the tridiagonal matrix T, and the elements above the first superdiagonal, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors; if UPLO = 'L', the diagonal and first subdiagonal of A are over- written by the corresponding elements of the tridiagonal matrix T, and the elements below the first subdiagonal, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

D Output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The diagonal elements of the tridiagonal matrix T: D(i) = A(i,i).

E Output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N-1). The off-diagonal elements of the tridiagonal matrix T: E(i) = A(i,i+1) if UPLO = 'U', E(i) = A(i+1,i) if UPLO = 'L'.

TAU Output parameter.

TAU is COMPLEX*16

TAU is an array, dimension (N-1). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= 1. For optimum performance LWORK >= N*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *chetrd*. It also exists with a native C interface as *LAPACKE_zhetrd*.

4.11.96 zhpev

zhpev computes all the eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix in packed storage.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhpev(JOBZ, UPLO, N, AP, W, Z, LDZ, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhpev_(const char *jobz, const char *uplo, const armpl_int_t *n,
            armpl_doublecomplex_t *ap, double *w, armpl_doublecomplex_t *z,
            const armpl_int_t *ldz, armpl_doublecomplex_t *work,
            double *rwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the Hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n.

On exit, AP is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the diagonal and first superdiagonal of the tridiagonal matrix T overwrite the corresponding elements of A, and if UPLO = 'L', the diagonal and first subdiagonal of T overwrite the corresponding elements of A.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the orthonormal eigenvectors of the matrix A, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (max(1, 2*N-1)) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (max(1, 3*N-2)) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *chpev*. It also exists with a native C interface as *LAPACKE_zhpev*.

4.11.97 zhpevd

zhpevd computes all the eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix A in packed storage. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhpevd(JOBZ, UPLO, N, AP, W, Z, LDZ, WORK, LWORK, RWORK, LRWORK,
                  IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhpevd_(const char *jobz, const char *uplo, const armpl_int_t *n,
             armpl_doublecomplex_t *ap, double *w, armpl_doublecomplex_t *z,
             const armpl_int_t *ldz, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, double *rwork,
             const armpl_int_t *lrwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the Hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n.

On exit, AP is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the diagonal and first superdiagonal of the tridiagonal matrix T overwrite the corresponding elements of A, and if UPLO = 'L', the diagonal and first subdiagonal of T overwrite the corresponding elements of A.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the orthonormal eigenvectors of the matrix A, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the required LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of array WORK. If N <= 1, LWORK must be at least 1. If JOBZ = 'N' and N > 1, LWORK must be at least N. If JOBZ = 'V' and N > 1, LWORK must be at least 2*N.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the required sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (MAX(1, LRWORK)). On exit, if INFO = 0, RWORK(1) returns the required LRWORK.

LRWORK Input parameter.

LRWORK is INTEGER

The dimension of array RWORK. If N <= 1, LRWORK must be at least 1. If JOBZ = 'N' and N > 1, LRWORK must be at least N. If JOBZ = 'V' and N > 1, LRWORK must be at least 1 + 5* N + 2*N**2.

If LRWORK = -1, then a workspace query is assumed; the routine only calculates the required sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the required LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of array IWORK. If JOBZ = 'N' or N <= 1, LIWORK must be at least 1. If JOBZ = 'V' and N > 1, LIWORK must be at least 3 + 5*N.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the required sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *chpevd*. It also exists with a native C interface as *LAPACKE_zhpevd*.

4.11.98 zhpevx

zhpevx computes selected eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix A in packed storage. Eigenvalues/vectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhpevx(JOBZ, RANGE, UPLO, N, AP, VL, VU, IL, IU, ABSTOL, M, W, Z,
                  LDZ, WORK, RWORK, IWORK, IFAIL, INFO)
```

C specification:

```
#include "armpl.h"

void zhpevx_(const char *jobz, const char *range, const char *uplo,
             const armpl_int_t *n, armpl_doublecomplex_t *ap,
             const double *vl, const double *vu, const armpl_int_t *il,
             const armpl_int_t *iu, const double *abstol, armpl_int_t *m,
             double *w, armpl_doublecomplex_t *z, const armpl_int_t *ldz,
             armpl_doublecomplex_t *work, double *rwork, armpl_int_t *iwork,
             armpl_int_t *ifail, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= ‘A’: all eigenvalues will be found; = ‘V’: all eigenvalues in the half-open interval (VL,VU] will be found; = ‘I’: the IL-th through IU-th eigenvalues will be found.

UPLO Input parameter.

UPLO is CHARACTER*1

= ‘U’: Upper triangle of A is stored; = ‘L’: Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the Hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = ‘U’, AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = ‘L’, AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n.

On exit, AP is overwritten by values generated during the reduction to tridiagonal form. If UPLO = ‘U’, the diagonal and first superdiagonal of the tridiagonal matrix T overwrite the corresponding elements of A, and if UPLO = ‘L’, the diagonal and first subdiagonal of T overwrite the corresponding elements of A.

VL Input parameter.

VL is DOUBLE PRECISION

If RANGE=’V’, the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = ‘A’ or ‘I’.

VU Input parameter.

VU is DOUBLE PRECISION

If RANGE=’V’, the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = ‘A’ or ‘I’.

IL Input parameter.

IL is INTEGER

If RANGE=’I’, the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = ‘A’ or ‘V’.

IU Input parameter.

IU is INTEGER

If RANGE=’I’, the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = ‘A’ or ‘V’.

ABSTOL Input parameter.

ABSTOL is DOUBLE PRECISION

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing AP to tridiagonal form.

Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*DLAMCH(‘S’), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*DLAMCH(‘S’).

See “Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy,” by Demmel and Kahan, LAPACK Working Note #3.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the selected eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). If an eigenvector fails to converge, then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL. If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (7*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (5*N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (N)

If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, then i eigenvectors failed to converge. Their indices are stored in array IFAIL.

Related Information

For this routine in other precisions, please see *chpevx*. It also exists with a native C interface as *LAPACKE_zhpevx*.

4.11.99 zhptrd

zhptrd reduces a complex Hermitian matrix A stored in packed form to real symmetric tridiagonal form T by a unitary similarity transformation: $Q^H * A * Q = T$.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhptrd(UPLO, N, AP, D, E, TAU, INFO)
```

C specification:

```
#include "armpl.h"

void zhptrd_(const char *uplo, const armpl_int_t *n,
             armpl_doublecomplex_t *ap, double *d, double *e,
             armpl_doublecomplex_t *tau, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the Hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n. On exit, if UPLO = 'U', the diagonal and first superdiagonal of A are overwritten by the corresponding elements of the tridiagonal matrix T, and the elements above the first superdiagonal, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors; if UPLO = 'L', the diagonal and first subdiagonal of A are over- written by the corresponding elements of the tridiagonal matrix T, and the elements below the first subdiagonal, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors. See Further Details.

D Output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The diagonal elements of the tridiagonal matrix T: D(i) = A(i,i).

E Output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N-1). The off-diagonal elements of the tridiagonal matrix T: E(i) = A(i,i+1) if UPLO = 'U', E(i) = A(i+1,i) if UPLO = 'L'.

TAU Output parameter.

TAU is COMPLEX*16

TAU is an array, dimension (N-1). The scalar factors of the elementary reflectors (see Further Details).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *chptrd*. It also exists with a native C interface as *LAPACKE_zhptrd*.

4.11.100 zpteqr

`zpteqr` computes all eigenvalues and, optionally, eigenvectors of a symmetric positive definite tridiagonal matrix by first factoring the matrix using DPTTRF and then calling ZBDSQR to compute the singular values of the bidiagonal factor.

This routine computes the eigenvalues of the positive definite tridiagonal matrix to high relative accuracy. This means that if the eigenvalues range over many orders of magnitude in size, then the small eigenvalues and corresponding eigenvectors will be computed more accurately than, for example, with the standard QR method.

The eigenvectors of a full or band positive definite Hermitian matrix can also be found if ZHETRD, ZHPTRD, or ZHBTRD has been used to reduce this matrix to tridiagonal form. (The reduction to tridiagonal form, however, may preclude the possibility of obtaining high relative accuracy in the small eigenvalues of the original matrix, if these eigenvalues range over many orders of magnitude.)

Syntax

Fortran specification:

```
use armpl_library

subroutine zpteqr(COMPZ, N, D, E, Z, LDZ, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zpteqr_(const char *compz, const armpl_int_t *n, double *d, double *e,
             armpl_doublecomplex_t *z, const armpl_int_t *ldz, double *work,
             armpl_int_t *info, ... );
```

Parameters

COMPZ Input parameter.

`COMPZ` is `CHARACTER*1`

= 'N': Compute eigenvalues only. = 'V': Compute eigenvectors of original Hermitian matrix also. Array `Z` contains the unitary matrix used to reduce the original matrix to tridiagonal form. = 'I': Compute eigenvectors of tridiagonal matrix also.

N Input parameter.

`N` is `INTEGER`

The order of the matrix. `N` >= 0.

D Input and output parameter.

`D` is `DOUBLE PRECISION`

`D` is an array, dimension (`N`). On entry, the n diagonal elements of the tridiagonal matrix. On normal exit, `D` contains the eigenvalues, in descending order.

E Input and output parameter.

`E` is `DOUBLE PRECISION`

E is an array, dimension (N-1). On entry, the (n-1) subdiagonal elements of the tridiagonal matrix. On exit, E has been destroyed.

Z Input and output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, N). On entry, if COMPZ = 'V', the unitary matrix used in the reduction to tridiagonal form. On exit, if COMPZ = 'V', the orthonormal eigenvectors of the original Hermitian matrix; if COMPZ = 'I', the orthonormal eigenvectors of the tridiagonal matrix. If INFO > 0 on exit, Z contains the eigenvectors associated with only the stored eigenvalues. If COMPZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if COMPZ = 'V' or 'I', LDZ >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (4*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, and i is: <= N the Cholesky factorization of the matrix could not be performed because the i-th principal minor was not positive definite. > N the SVD algorithm failed to converge; if INFO = N+i, i off-diagonal elements of the bidiagonal factor did not converge to zero.

Related Information

For this routine in other precisions, please see *cpteqr*, *dpteqr* and *spteqr*. It also exists with a native C interface as *LAPACKE_zpteqr*.

4.11.101 zstedc

zstedc computes all eigenvalues and, optionally, eigenvectors of a symmetric tridiagonal matrix using the divide and conquer method. The eigenvectors of a full or band complex Hermitian matrix can also be found if ZHETRD or ZHPTRD or ZHBTRD has been used to reduce this matrix to tridiagonal form.

This code makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none. See DLAED3 for details.

Syntax

Fortran specification:

```
use armpl_library

subroutine zstedc(COMPZ, N, D, E, Z, LDZ, WORK, LWORK, RWORK, LRWORK, IWORK,
                  LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zstedc_(const char *compz, const armpl_int_t *n, double *d, double *e,
             armpl_doublecomplex_t *z, const armpl_int_t *ldz,
             armpl_doublecomplex_t *work, const armpl_int_t *lwork,
             double *rwork, const armpl_int_t *lrwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

COMPZ Input parameter.

COMPZ is CHARACTER*1

= 'N': Compute eigenvalues only. = 'I': Compute eigenvectors of tridiagonal matrix also. = 'V': Compute eigenvectors of original Hermitian matrix also. On entry, Z contains the unitary matrix used to reduce the original matrix to tridiagonal form.

N Input parameter.

N is INTEGER

The dimension of the symmetric tridiagonal matrix. N >= 0.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry, the diagonal elements of the tridiagonal matrix. On exit, if INFO = 0, the eigenvalues in ascending order.

E Input and output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N-1). On entry, the subdiagonal elements of the tridiagonal matrix. On exit, E has been destroyed.

Z Input and output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, N). On entry, if COMPZ = 'V', then Z contains the unitary matrix used in the reduction to tridiagonal form. On exit, if INFO = 0, then if COMPZ = 'V', Z contains the orthonormal eigenvectors of the original Hermitian matrix, and if COMPZ = 'I', Z contains the orthonormal eigenvectors of the symmetric tridiagonal matrix. If COMPZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1. If eigenvectors are desired, then LDZ >= max(1, N).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If COMPZ = 'N' or 'I', or N <= 1, LWORK must be at least 1. If COMPZ = 'V' and N > 1, LWORK must be at least N*N. Note that for COMPZ = 'V', then if N is less than or equal to the minimum divide size, usually 25, then LWORK need only be 1.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (MAX(1, LRWORK)). On exit, if INFO = 0, RWORK(1) returns the optimal LRWORK.

LRWORK Input parameter.

LRWORK is INTEGER

The dimension of the array RWORK. If COMPZ = 'N' or N <= 1, LRWORK must be at least 1. If COMPZ = 'V' and N > 1, LRWORK must be at least 1 + 3* N + 2*N*lg N + 4*N**2 , where lg(N) = smallest integer k such that 2**k >= N. If COMPZ = 'I' and N > 1, LRWORK must be at least 1 + 4* N + 2*N**2 . Note that for COMPZ = 'I' or 'V', then if N is less than or equal to the minimum divide size, usually 25, then LRWORK need only be max(1,2*(N-1)).

If LRWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. If COMPZ = 'N' or N <= 1, LIWORK must be at least 1. If COMPZ = 'V' or N > 1, LIWORK must be at least 6 + 6* N + 5*N*lg N. If COMPZ = 'I' or N > 1, LIWORK must be at least 3 + 5* N . Note that for COMPZ = 'I' or 'V', then if N is less than or equal to the minimum divide size, usually 25, then LIWORK need only be 1.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: The algorithm failed to compute an eigenvalue while working on the submatrix lying in rows and columns INFO/(N+1) through mod(INFO,N+1).

Related Information

For this routine in other precisions, please see *cstedc*, *dstedc* and *sstedc*. It also exists with a native C interface as *LAPACKE_zstedc*.

4.11.102 zstegr

zstegr computes selected eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix T. Any such unreduced matrix has a well defined set of pairwise different real eigenvalues, the corresponding real eigenvectors are pairwise orthogonal.

The spectrum may be computed either completely or partially by specifying either an interval (VL,VU] or a range of indices IL:IU for the desired eigenvalues.

zstegr is a compatibility wrapper around the improved ZSTEMR routine. See DSTEMR for further details.

One important change is that the ABSTOL parameter no longer provides any benefit and hence is no longer used.

Note : zstegr and ZSTEMR work only on machines which follow IEEE-754 floating-point standard in their handling of infinities and NaNs. Normal execution may create these exceptiona values and hence may abort due to a floating point exception in environments which do not conform to the IEEE-754 standard.

Syntax

Fortran specification:

```
use armpl_library

subroutine zstegr(JOBZ, RANGE, N, D, E, VL, VU, IL, IU, ABSTOL, M, W, Z, LDZ,
                  ISUPPZ, WORK, LWORK, IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zstegr_(const char *jobz, const char *range, const armpl_int_t *n,
             double *d, double *e, const double *vl, const double *vu,
             const armpl_int_t *il, const armpl_int_t *iu,
             const double *abstol, armpl_int_t *m, double *w,
             armpl_doublecomplex_t *z, const armpl_int_t *ldz,
             armpl_int_t *isuppz, double *work, const armpl_int_t *lwork,
             armpl_int_t *iwork, const armpl_int_t *liwork, armpl_int_t *info,
             ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found. = 'V': all eigenvalues in the half-open interval (VL,VU] will be found. = 'I': the IL-th through IU-th eigenvalues will be found.

N Input parameter.

N is INTEGER

The order of the matrix. N >= 0.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry, the N diagonal elements of the tridiagonal matrix T. On exit, D is overwritten.

E Input and output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N). On entry, the (N-1) subdiagonal elements of the tridiagonal matrix T in elements 1 to N-1 of E. E(N) need not be set on input, but is used internally as workspace. On exit, E is overwritten.

VL Input parameter.

VL is DOUBLE PRECISION

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is DOUBLE PRECISION

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is DOUBLE PRECISION

Unused. Was the absolute error tolerance for the eigenvalues/eigenvectors in previous versions.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). The first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', and if INFO = 0, then the first M columns of Z contain the orthonormal eigenvectors of the matrix T corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used. Supplying N columns is always safe.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', then LDZ >= max(1, N).

ISUPPZ Output parameter.

ISUPPZ is INTEGER array, dimension (2*max(1, M))

The support of the eigenvectors in Z, i.e., the indices indicating the nonzero elements in Z. The i-th computed eigenvector is nonzero only in elements ISUPPZ(2*i-1) through ISUPPZ(2*i). This is relevant in the case when the matrix is split. ISUPPZ is only accessed when JOBZ is 'V' and N > 0.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (LWORK). On exit, if INFO = 0, WORK(1) returns the optimal (and minimal) LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,18*N) if JOBZ = 'V', and LWORK >= max(1,12*N) if JOBZ = 'N'. If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (LIWORK)

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. LIWORK >= max(1,10*N) if the eigenvectors are desired, and LIWORK >= max(1,8*N) if only the eigenvalues are to be computed. If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the IWORK array, returns this value as the first entry of the IWORK array, and no error message related to LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

On exit, INFO = 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = 1X, internal error in DLARRE, if INFO = 2X, internal error in ZLARRV. Here, the digit X = ABS(IINFO) < 10, where IINFO is the nonzero error code returned by DLARRE or ZLARRV, respectively.

Related Information

For this routine in other precisions, please see *cstegr*, *dstegr* and *sstegr*. It also exists with a native C interface as *LAPACKE_zstegr*.

4.11.103 zstein

zstein computes the eigenvectors of a real symmetric tridiagonal matrix T corresponding to specified eigenvalues, using inverse iteration.

The maximum number of iterations allowed for each eigenvector is specified by an internal parameter MAXITS (currently set to 5).

Although the eigenvectors are real, they are stored in a complex array, which may be passed to ZUNMTR or ZUPMTR for back transformation to the eigenvectors of a complex Hermitian matrix which was reduced to tridiagonal form.

Syntax

Fortran specification:

```
use armpl_library

subroutine zstein(N, D, E, M, W, IBLOCK, ISPLIT, Z, LDZ, WORK, IWORK, IFAIL,
                  INFO)
```

C specification:

```
#include "armpl.h"

void zstein_(const armpl_int_t *n, const double *d, const double *e,
             const armpl_int_t *m, const double *w, const armpl_int_t *iblock,
             const armpl_int_t *isplit, armpl_doublecomplex_t *z,
             const armpl_int_t *ldz, double *work, armpl_int_t *iwork,
             armpl_int_t *ifail, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix. N >= 0.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The n diagonal elements of the tridiagonal matrix T.

E Input parameter.

E is DOUBLE PRECISION

E is an array, dimension (N-1). The (n-1) subdiagonal elements of the tridiagonal matrix T, stored in elements 1 to N-1.

M Input parameter.

M is INTEGER

The number of eigenvectors to be found. 0 <= M <= N.

W Input parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). The first M elements of W contain the eigenvalues for which eigenvectors are to be computed. The eigenvalues should be grouped by split-off block and ordered from smallest to largest within the block. (The output array W from DSTEBZ with ORDER = 'B' is expected here.)

IBLOCK Input parameter.

IBLOCK is INTEGER array, dimension (N)

The submatrix indices associated with the corresponding eigenvalues in W; IBLOCK(i)=1 if eigenvalue W(i) belongs to the first submatrix from the top, =2 if W(i) belongs to the second submatrix, etc. (The output array IBLOCK from DSTEBZ is expected here.)

ISPLIT Input parameter.

ISPLIT is INTEGER array, dimension (N)

The splitting points, at which T breaks up into submatrices. The first submatrix consists of rows/columns 1 to ISPLIT(1), the second of rows/columns ISPLIT(1)+1 through ISPLIT(2), etc. (The output array ISPLIT from DSTEBZ is expected here.)

Z Output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, M). The computed eigenvectors. The eigenvector associated with the eigenvalue W(i) is stored in the i-th column of Z. Any vector which fails to converge is set to its current iterate after MAXITS iterations. The imaginary parts of the eigenvectors are set to zero.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (5*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (M)

On normal exit, all elements of IFAIL are zero. If one or more eigenvectors fail to converge after MAXITS iterations, then their indices are stored in array IFAIL.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, then i eigenvectors failed to converge in MAXITS iterations. Their indices are stored in array IFAIL.

Related Information

For this routine in other precisions, please see *cstein*, *dstein* and *sstein*. It also exists with a native C interface as *LAPACKE_zstein*.

4.11.104 zstemr

zstemr computes selected eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix T. Any such unreduced matrix has a well defined set of pairwise different real eigenvalues, the corresponding real eigenvectors are pairwise orthogonal.

The spectrum may be computed either completely or partially by specifying either an interval (VL,VU] or a range of indices IL:IU for the desired eigenvalues.

Depending on the number of desired eigenvalues, these are computed either by bisection or the dqds algorithm. Numerically orthogonal eigenvectors are computed by the use of various suitable L D L^T factorizations near clusters of close eigenvalues (referred to as RRRs, Relatively Robust Representations). An informal sketch of the algorithm follows.

For each unreduced block (submatrix) of T,

```
(a) Compute T - sigma I  = L D L^T, so that L and D
    define all the wanted eigenvalues to high relative accuracy.
    This means that small relative changes in the entries of D and L
    cause only small relative changes in the eigenvalues and
    eigenvectors. The standard (unfactored) representation of the
    tridiagonal matrix T does not have this property in general.
(b) Compute the eigenvalues to suitable accuracy.
    If the eigenvectors are desired, the algorithm attains full
    accuracy of the computed eigenvalues only right before
    the corresponding vectors have to be computed, see steps c) and d).
(c) For each cluster of close eigenvalues, select a new
    shift close to the cluster, find a new factorization, and refine
    the shifted eigenvalues to suitable accuracy.
(d) For each eigenvalue with a large enough relative separation compute
```

(continues on next page)

(continued from previous page)

```
    the corresponding eigenvector by forming a rank revealing twisted
    factorization. Go back to (c) for any clusters that remain.
```

For more details, see: - Inderjit S. Dhillon and Beresford N. Parlett: “Multiple representations

```
 to compute orthogonal eigenvectors of symmetric tridiagonal matrices,"
Linear Algebra and its Applications, 387(1), pp. 1-28, August 2004.
```

- Inderjit Dhillon and Beresford Parlett: “Orthogonal Eigenvectors and

```
 Relative Gaps," SIAM Journal on Matrix Analysis and Applications, Vol. 25,
2004.  Also LAPACK Working Note 154.
```

- Inderjit Dhillon: “A new O(n^2) algorithm for the symmetric

```
 tridiagonal eigenvalue/eigenvector problem",
Computer Science Division Technical Report No. UCB/CSD-97-971,
UC Berkeley, May 1997.
```

Further Details 1.‘‘zstemr‘‘ works only on machines which follow IEEE-754 floating-point standard in their handling of infinities and NaNs. This permits the use of efficient inner loops avoiding a check for zero divisors.

2. LAPACK routines can be used to reduce a complex Hermitean matrix to real symmetric tridiagonal form.

(Any complex Hermitean tridiagonal matrix has real values on its diagonal and potentially complex numbers on its off-diagonals. By applying a similarity transform with an appropriate diagonal matrix diag(1,e^{i phy_1}, … , e^{i phy_{n-1}}), the complex Hermitean matrix can be transformed into a real symmetric matrix and complex arithmetic can be entirely avoided.)

While the eigenvectors of the real symmetric tridiagonal matrix are real, the eigenvectors of original complex Hermitean matrix have complex entries in general. Since LAPACK drivers overwrite the matrix data with the eigenvectors, `zstemr` accepts complex workspace to facilitate interoperability with ZUNMTR or ZUPMTR.

Syntax

Fortran specification:

```
use armpl_library

subroutine zstemr(JOBZ, RANGE, N, D, E, VL, VU, IL, IU, M, W, Z, LDZ, NZC,
                  ISUPPZ, TRYRAC, WORK, LWORK, IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zstemr_(const char *jobz, const char *range, const armpl_int_t *n,
             double *d, double *e, const double *vl, const double *vu,
             const armpl_int_t *il, const armpl_int_t *iu, armpl_int_t *m,
             double *w, armpl_doublecomplex_t *z, const armpl_int_t *ldz,
             const armpl_int_t *nzc, armpl_int_t *isuppz, armpl_int_t *tryrac,
             double *work, const armpl_int_t *lwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

`JOBZ` is `CHARACTER*1`

= ‘`N`’: Compute eigenvalues only; = ‘V’: Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found. = 'V': all eigenvalues in the half-open interval (VL,VU] will be found. = 'I': the IL-th through IU-th eigenvalues will be found.

N Input parameter.

N is INTEGER

The order of the matrix. N >= 0.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry, the N diagonal elements of the tridiagonal matrix T. On exit, D is overwritten.

E Input and output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N). On entry, the (N-1) subdiagonal elements of the tridiagonal matrix T in elements 1 to N-1 of E. E(N) need not be set on input, but is used internally as workspace. On exit, E is overwritten.

VL Input parameter.

VL is DOUBLE PRECISION

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is DOUBLE PRECISION

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0. Not referenced if RANGE = 'A' or 'V'.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). The first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', and if INFO = 0, then the first M columns of Z contain the orthonormal eigenvectors of the matrix T corresponding to the selected eigenvalues, with the i-th

column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and can be computed with a workspace query by setting NZC = -1, see below.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', then LDZ >= max(1, N).

NZC Input parameter.

NZC is INTEGER

The number of eigenvectors to be held in the array Z. If RANGE = 'A', then NZC >= max(1, N). If RANGE = 'V', then NZC >= the number of eigenvalues in (VL,VU]. If RANGE = 'I', then NZC >= IU-IL+1. If NZC = -1, then a workspace query is assumed; the routine calculates the number of columns of the array Z that are needed to hold the eigenvectors. This value is returned as the first entry of the Z array, and no error message related to NZC is issued by XERBLA.

ISUPPZ Output parameter.

ISUPPZ is INTEGER array, dimension (2*max(1, M))

The support of the eigenvectors in Z, i.e., the indices indicating the nonzero elements in Z. The i-th computed eigenvector is nonzero only in elements ISUPPZ(2*i-1) through ISUPPZ(2*i). This is relevant in the case when the matrix is split. ISUPPZ is only accessed when JOBZ is 'V' and N > 0.

TRYRAC Input and output parameter.

TRYRAC is LOGICAL

If TRYRAC.EQ..TRUE., indicates that the code should check whether the tridiagonal matrix defines its eigenvalues to high relative accuracy. If so, the code uses relative-accuracy preserving algorithms that might be (a bit) slower depending on the matrix. If the matrix does not define its eigenvalues to high relative accuracy, the code can uses possibly faster algorithms. If TRYRAC.EQ..FALSE., the code is not required to guarantee relatively accurate eigenvalues and can use the fastest possible techniques. On exit, a .TRUE. TRYRAC will be set to .FALSE. if the matrix does not define its eigenvalues to high relative accuracy.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (LWORK). On exit, if INFO = 0, WORK(1) returns the optimal (and minimal) LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,18*N) if JOBZ = 'V', and LWORK >= max(1,12*N) if JOBZ = 'N'. If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (LIWORK)

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. LIWORK >= max(1,10*N) if the eigenvectors are desired, and LIWORK >= max(1,8*N) if only the eigenvalues are to be computed. If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the IWORK array, returns this value as the first entry of the IWORK array, and no error message related to LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

On exit, INFO = 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = 1X, internal error in DLARRE, if INFO = 2X, internal error in ZLARRV. Here, the digit X = ABS(IINFO) < 10, where IINFO is the nonzero error code returned by DLARRE or ZLARRV, respectively.

Related Information

For this routine in other precisions, please see *cstemr*, *dstemr* and *sstemr*. It also exists with a native C interface as *LAPACKE_zstemr*.

4.11.105 zsteqr

zsteqr computes all eigenvalues and, optionally, eigenvectors of a symmetric tridiagonal matrix using the implicit QL or QR method. The eigenvectors of a full or band complex Hermitian matrix can also be found if ZHETRD or ZHPTRD or ZHBTRD has been used to reduce this matrix to tridiagonal form.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsteqr(COMPZ, N, D, E, Z, LDZ, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zsteqr_(const char *compz, const armpl_int_t *n, double *d, double *e,
             armpl_doublecomplex_t *z, const armpl_int_t *ldz, double *work,
             armpl_int_t *info, ... );
```

Parameters

COMPZ Input parameter.

COMPZ is CHARACTER*1

= 'N': Compute eigenvalues only. = 'V': Compute eigenvalues and eigenvectors of the original Hermitian matrix. On entry, Z must contain the unitary matrix used to reduce the original matrix to tridiagonal form. = 'I': Compute eigenvalues and eigenvectors of the tridiagonal matrix. Z is initialized to the identity matrix.

N Input parameter.

N is INTEGER

The order of the matrix. N >= 0.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry, the diagonal elements of the tridiagonal matrix. On exit, if INFO = 0, the eigenvalues in ascending order.

E Input and output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N-1). On entry, the (n-1) subdiagonal elements of the tridiagonal matrix. On exit, E has been destroyed.

Z Input and output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, N). On entry, if COMPZ = 'V', then Z contains the unitary matrix used in the reduction to tridiagonal form. On exit, if INFO = 0, then if COMPZ = 'V', Z contains the orthonormal eigenvectors of the original Hermitian matrix, and if COMPZ = 'I', Z contains the orthonormal eigenvectors of the symmetric tridiagonal matrix. If COMPZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if eigenvectors are desired, then LDZ >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (max(1,2*N-2)). If COMPZ = 'N', then WORK is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: the algorithm has failed to find all the eigenvalues in a total of 30* N iterations; if INFO = i, then i elements of E have not converged to zero; on exit, D and E contain the elements of a symmetric tridiagonal matrix which is unitarily similar to the original matrix.

Related Information

For this routine in other precisions, please see *csteqr*, *dsteqr* and *ssteqr*. It also exists with a native C interface as *LAPACKE_zsteqr*.

4.11.106 zungtr

zungtr generates a complex unitary matrix Q which is defined as the product of n-1 elementary reflectors of order N, as returned by ZHETRD:

if UPLO = 'U', Q = H(n-1) . . . H(2) H(1),

if UPLO = 'L', Q = H(1) H(2) . . . H(n-1).

Syntax

Fortran specification:

```
use armpl_library

subroutine zungtr(UPLO, N, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zungtr_(const char *uplo, const armpl_int_t *n, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, const armpl_doublecomplex_t *tau,
             armpl_doublecomplex_t *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A contains elementary reflectors from ZHETRD; = 'L': Lower triangle of A contains elementary reflectors from ZHETRD.

N Input parameter.

N is INTEGER

The order of the matrix Q. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the vectors which define the elementary reflectors, as returned by ZHETRD. On exit, the N-by-N unitary matrix Q.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= N.

TAU Input parameter.

TAU is COMPLEX*16

TAU is an array, dimension (N-1). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by ZHETRD.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= N-1. For optimum performance LWORK >= (N-1)*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cungtr*. It also exists with a native C interface as *LAPACKE_zungtr*.

4.11.107 zunm22

```
ZUNM22 overwrites the general complex M-by-N matrix C with

                 SIDE = 'L'     SIDE = 'R'
 TRANS = 'N':      Q * C          C * Q
 TRANS = 'C':      Q**H * C       C * Q**H

 where Q is a complex unitary matrix of order NQ, with NQ = M if
 SIDE = 'L' and NQ = N if SIDE = 'R'.
 The unitary matrix Q processes a 2-by-2 block structure

        [  Q11  Q12  ]
    Q = [            ]
        [  Q21  Q22  ],

 where Q12 is an N1-by-N1 lower triangular matrix and Q21 is an
 N2-by-N2 upper triangular matrix.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zunm22(SIDE, TRANS, M, N, N1, N2, Q, LDQ, C, LDC, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void zunm22_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *n1,
             const armpl_int_t *n2, const armpl_doublecomplex_t *q,
             const armpl_int_t *ldq, armpl_doublecomplex_t *c,
             const armpl_int_t *ldc, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left; = 'R': apply Q or Q^H from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply Q (No transpose); = 'C': apply Q^H (Conjugate transpose).

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

N1 Input parameter.

N2 Input parameter.

N1 is INTEGER

N2 is INTEGER The dimension of Q12 and Q21, respectively. N1, N2 >= 0. The following requirement must be satisfied: N1 + N2 = M if SIDE = 'L' and N1 + N2 = N if SIDE = 'R'.

Q Input parameter.

Q is COMPLEX*16

Q is an array, dimension. (LDQ, M) if SIDE = 'L' (LDQ, N) if SIDE = 'R'

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, M) if SIDE = 'L'; LDQ >= max(1, N) if SIDE = 'R'.

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^H * C or C*Q^H or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, N); if SIDE = 'R', LWORK >= max(1, M). For optimum performance LWORK >= M*N.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cunm22*.

4.11.108 zunmtr

zunmtr overwrites the general complex M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'C': Q^H * C C * Q^H

where Q is a complex unitary matrix of order nq, with nq = m if SIDE = 'L' and nq = n if SIDE = 'R'. Q is defined as the product of nq-1 elementary reflectors, as returned by ZHETRD:

if UPLO = 'U', Q = H(nq-1) . . . H(2) H(1);

if UPLO = 'L', Q = H(1) H(2) . . . H(nq-1).

Syntax

Fortran specification:

```
use armpl_library

subroutine zunmtr(SIDE, UPLO, TRANS, M, N, A, LDA, TAU, C, LDC, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void zunmtr_(const char *side, const char *uplo, const char *trans,
             const armpl_int_t *m, const armpl_int_t *n,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_doublecomplex_t *tau, armpl_doublecomplex_t *c,
             const armpl_int_t *ldc, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left; = 'R': apply Q or Q^H from the Right.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A contains elementary reflectors from ZHETRD; = 'L': Lower triangle of A contains elementary reflectors from ZHETRD.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'C': Conjugate transpose, apply Q^H .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension. (LDA, M) if SIDE = 'L' (LDA, N) if SIDE = 'R' The vectors which define the elementary reflectors, as returned by ZHETRD.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M) if SIDE = 'L'; LDA >= max(1, N) if SIDE = 'R'.

TAU Input parameter.

TAU is COMPLEX*16

TAU is an array, dimension. (M-1) if SIDE = 'L' (N-1) if SIDE = 'R' TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by ZHETRD.

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^H * C or C*Q^H or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, N); if SIDE = 'R', LWORK >= max(1, M). For optimum performance LWORK >= N*NB if SIDE = 'L', and LWORK >=M*NB if SIDE = 'R', where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cunmtr*. It also exists with a native C interface as *LAPACKE_zunmtr*.

4.11.109 zupgtr

zupgtr generates a complex unitary matrix Q which is defined as the product of n-1 elementary reflectors H(i) of order n, as returned by ZHPTRD using packed storage:

if UPLO = 'U', Q = H(n-1) . . . H(2) H(1),

if UPLO = 'L', Q = H(1) H(2) . . . H(n-1).

Syntax

Fortran specification:

```
use armpl_library

subroutine zupgtr(UPLO, N, AP, TAU, Q, LDQ, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zupgtr_(const char *uplo, const armpl_int_t *n,
             const armpl_doublecomplex_t *ap,
             const armpl_doublecomplex_t *tau, armpl_doublecomplex_t *q,
             const armpl_int_t *ldq, armpl_doublecomplex_t *work,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangular packed storage used in previous call to ZHPTRD; = 'L': Lower triangular packed storage used in previous call to ZHPTRD.

N Input parameter.

N is INTEGER

The order of the matrix Q. N >= 0.

AP Input parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). The vectors which define the elementary reflectors, as returned by ZHPTRD.

TAU Input parameter.

TAU is COMPLEX*16

TAU is an array, dimension (N-1). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by ZHPTRD.

Q Output parameter.

Q is COMPLEX*16

Q is an array, dimension (LDQ, N). The N-by-N unitary matrix Q.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, N).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N-1) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cupgtr*. It also exists with a native C interface as *LAPACKE_zupgtr*.

4.11.110 zupmtr

zupmtr overwrites the general complex M-by-N matrix C with

```
SIDE = 'L'      SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'C': Q^H * C C * Q^H

where Q is a complex unitary matrix of order nq, with nq = m if SIDE = 'L' and nq = n if SIDE = 'R'. Q is defined as the product of nq-1 elementary reflectors, as returned by ZHPTRD using packed storage:

if UPLO = 'U', Q = H(nq-1) . . . H(2) H(1);

if UPLO = 'L', Q = H(1) H(2) . . . H(nq-1).

Syntax

Fortran specification:

```
use armpl_library

subroutine zupmtr(SIDE, UPLO, TRANS, M, N, AP, TAU, C, LDC, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zupmtr_(const char *side, const char *uplo, const char *trans,
             const armpl_int_t *m, const armpl_int_t *n,
             const armpl_doublecomplex_t *ap,
             const armpl_doublecomplex_t *tau, armpl_doublecomplex_t *c,
             const armpl_int_t *ldc, armpl_doublecomplex_t *work,
             armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left; = 'R': apply Q or Q^H from the Right.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangular packed storage used in previous call to ZHPTRD; = 'L': Lower triangular packed storage used in previous call to ZHPTRD.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'C': Conjugate transpose, apply Q^H .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

AP Input parameter.

AP is COMPLEX*16

AP is an array, dimension. (M*(M+1)/2) if SIDE = 'L' (N*(N+1)/2) if SIDE = 'R' The vectors which define the elementary reflectors, as returned by ZHPTRD. AP is modified by the routine but restored on exit.

TAU Input parameter.

TAU is COMPLEX*16

TAU is an array, dimension (M-1) if SIDE = 'L'. or (N-1) if SIDE = 'R' TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by ZHPTRD.

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^H * C or C*Q^H or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension. (N) if SIDE = 'L' (M) if SIDE = 'R'

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cupmtr*. It also exists with a native C interface as *LAPACKE_zupmtr*.

4.12 LAPACK generalized symmetric eigenvalues routines

4.12.1 chbgst

chbgst reduces a complex Hermitian-definite banded generalized eigenproblem A*x = lambda*B*x to standard form C*y = lambda*y, such that C has the same bandwidth as A.

B must have been previously factorized as S^H *S by CPBSTF, using a split Cholesky factorization. A is overwritten by C = X^H *A*X, where X = S**(-1)*Q and Q is a unitary matrix chosen to preserve the bandwidth of A.

Syntax

Fortran specification:

```
use armpl_library

subroutine chbgst(VECT, UPLO, N, KA, KB, AB, LDAB, BB, LDBB, X, LDX, WORK,
                  RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void chbgst_(const char *vect, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *ka, const armpl_int_t *kb,
             armpl_singlecomplex_t *ab, const armpl_int_t *ldab,
             const armpl_singlecomplex_t *bb, const armpl_int_t *ldbb,
             armpl_singlecomplex_t *x, const armpl_int_t *ldx,
             armpl_singlecomplex_t *work, float *rwork, armpl_int_t *info,
             ... );
```

Parameters

VECT Input parameter.

VECT is CHARACTER*1

= 'N': do not form the transformation matrix X; = 'V': form X.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

KA Input parameter.

KA is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KA >= 0.

KB Input parameter.

KB is INTEGER

The number of superdiagonals of the matrix B if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KA >= KB >= 0.

AB Input and output parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the Hermitian band matrix A, stored in the first ka+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(ka+1+i-j,j) = A(i,j) for max(1,j-ka)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+ka).

On exit, the transformed matrix X^H *A*X, stored in the same format as A.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KA+1.

BB Input parameter.

BB is COMPLEX

BB is an array, dimension (LDBB, N). The banded factor S from the split Cholesky factorization of B, as returned by CPBSTF, stored in the first kb+1 rows of the array.

LDBB Input parameter.

LDBB is INTEGER

The leading dimension of the array BB. LDBB >= KB+1.

X Output parameter.

X is COMPLEX

X is an array, dimension (LDX, N). If VECT = 'V', the n-by-n matrix X. If VECT = 'N', the array X is not referenced.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N) if VECT = 'V'; LDX >= 1 otherwise.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *zhbgst*. It also exists with a native C interface as *LAPACKE_chbgst*.

4.12.2 chbgv

chbgv computes all the eigenvalues, and optionally, the eigenvectors of a complex generalized Hermitian-definite banded eigenproblem, of the form A*x=(lambda)*B*x. Here A and B are assumed to be Hermitian and banded, and B is also positive definite.

Syntax

Fortran specification:

```
use armpl_library

subroutine chbgv(JOBZ, UPLO, N, KA, KB, AB, LDAB, BB, LDBB, W, Z, LDZ, WORK,
                 RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void chbgv_(const char *jobz, const char *uplo, const armpl_int_t *n,
            const armpl_int_t *ka, const armpl_int_t *kb,
            armpl_singlecomplex_t *ab, const armpl_int_t *ldab,
            armpl_singlecomplex_t *bb, const armpl_int_t *ldbb, float *w,
            armpl_singlecomplex_t *z, const armpl_int_t *ldz,
            armpl_singlecomplex_t *work, float *rwork, armpl_int_t *info,
            ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangles of A and B are stored; = 'L': Lower triangles of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

KA Input parameter.

KA is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KA >= 0.

KB Input parameter.

KB is INTEGER

The number of superdiagonals of the matrix B if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KB >= 0.

AB Input and output parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the Hermitian band matrix A, stored in the first ka+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(ka+1+i-j,j) = A(i,j) for max(1,j-ka)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+ka).

On exit, the contents of AB are destroyed.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KA+1.

BB Input and output parameter.

BB is COMPLEX

BB is an array, dimension (LDBB, N). On entry, the upper or lower triangle of the Hermitian band matrix B, stored in the first kb+1 rows of the array. The j-th column of B is stored in the j-th column of the array BB as follows: if UPLO = 'U', BB(kb+1+i-j,j) = B(i,j) for max(1,j-kb)<=i<=j; if UPLO = 'L', BB(1+i-j,j) = B(i,j) for j<=i<=min(n,j+kb).

On exit, the factor S from the split Cholesky factorization B = S^H *S, as returned by CPBSTF.

LDBB Input parameter.

LDBB is INTEGER

The leading dimension of the array BB. LDBB >= KB+1.

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the matrix Z of eigenvectors, with the i-th column of Z holding the eigenvector associated with W(i). The eigenvectors are normalized so that Z^H *B* Z = I. If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= N.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (3*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is: <= N: the algorithm failed to converge: i off-diagonal elements of an intermediate tridiagonal form did not converge to zero; > N: if INFO = N + i, for 1 <= i <= N, then CPBSTF returned INFO = i: B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *zhbgv*. It also exists with a native C interface as *LAPACKE_chbgv*.

4.12.3 chbgvd

chbgvd computes all the eigenvalues, and optionally, the eigenvectors of a complex generalized Hermitian-definite banded eigenproblem, of the form A*x=(lambda)*B*x. Here A and B are assumed to be Hermitian and banded, and B is also positive definite. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine chbgvd(JOBZ, UPLO, N, KA, KB, AB, LDAB, BB, LDBB, W, Z, LDZ, WORK,
                  LWORK, RWORK, LRWORK, IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void chbgvd_(const char *jobz, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *ka, const armpl_int_t *kb,
             armpl_singlecomplex_t *ab, const armpl_int_t *ldab,
             armpl_singlecomplex_t *bb, const armpl_int_t *ldbb, float *w,
             armpl_singlecomplex_t *z, const armpl_int_t *ldz,
             armpl_singlecomplex_t *work, const armpl_int_t *lwork,
             float *rwork, const armpl_int_t *lrwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangles of A and B are stored; = 'L': Lower triangles of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

KA Input parameter.

KA is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KA >= 0.

KB Input parameter.

KB is INTEGER

The number of superdiagonals of the matrix B if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KB >= 0.

AB Input and output parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the Hermitian band matrix A, stored in the first ka+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(ka+1+i-j,j) = A(i,j) for max(1,j-ka)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+ka).

On exit, the contents of AB are destroyed.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KA+1.

BB Input and output parameter.

BB is COMPLEX

BB is an array, dimension (LDBB, N). On entry, the upper or lower triangle of the Hermitian band matrix B, stored in the first kb+1 rows of the array. The j-th column of B is stored in the j-th column of the array BB as follows: if UPLO = 'U', BB(kb+1+i-j,j) = B(i,j) for max(1,j-kb)<=i<=j; if UPLO = 'L', BB(1+i-j,j) = B(i,j) for j<=i<=min(n,j+kb).

On exit, the factor S from the split Cholesky factorization B = S^H *S, as returned by CPBSTF.

LDBB Input parameter.

LDBB is INTEGER

The leading dimension of the array BB. LDBB >= KB+1.

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the matrix Z of eigenvectors, with the i-th column of Z holding the eigenvector associated with W(i). The eigenvectors are normalized so that Z^H *B* Z = I. If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= N.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO=0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If N <= 1, LWORK >= 1. If JOBZ = 'N' and N > 1, LWORK >= N. If JOBZ = 'V' and N > 1, LWORK >= 2*N**2.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (MAX(1, LRWORK)). On exit, if INFO=0, RWORK(1) returns the optimal LRWORK.

LRWORK Input parameter.

LRWORK is INTEGER

The dimension of array RWORK. If N <= 1, LRWORK >= 1. If JOBZ = 'N' and N > 1, LRWORK >= N. If JOBZ = 'V' and N > 1, LRWORK >= 1 + 5* N + 2*N**2.

If LRWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO=0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of array IWORK. If JOBZ = 'N' or N <= 1, LIWORK >= 1. If JOBZ = 'V' and N > 1, LIWORK >= 3 + 5*N.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is: <= N: the algorithm failed to converge: i off-diagonal elements of an intermediate tridiagonal form did not converge to zero; > N: if INFO = N + i, for 1 <= i <= N, then CPBSTF returned INFO = i: B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *zhbgvd*. It also exists with a native C interface as *LAPACKE_chbgvd*.

4.12.4 chbgvx

chbgvx computes all the eigenvalues, and optionally, the eigenvectors of a complex generalized Hermitian-definite banded eigenproblem, of the form A*x=(lambda)*B*x. Here A and B are assumed to be Hermitian and banded, and B is also positive definite. Eigenvalues and eigenvectors can be selected by specifying either all eigenvalues, a range of values or a range of indices for the desired eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine chbgvx(JOBZ, RANGE, UPLO, N, KA, KB, AB, LDAB, BB, LDBB, Q, LDQ,
                  VL, VU, IL, IU, ABSTOL, M, W, Z, LDZ, WORK, RWORK, IWORK,
                  IFAIL, INFO)
```

C specification:

```
#include "armpl.h"

void chbgvx_(const char *jobz, const char *range, const char *uplo,
             const armpl_int_t *n, const armpl_int_t *ka,
             const armpl_int_t *kb, armpl_singlecomplex_t *ab,
             const armpl_int_t *ldab, armpl_singlecomplex_t *bb,
             const armpl_int_t *ldbb, armpl_singlecomplex_t *q,
             const armpl_int_t *ldq, const float *vl, const float *vu,
             const armpl_int_t *il, const armpl_int_t *iu,
```

(continues on next page)

(continued from previous page)

```
                const float *abstol, armpl_int_t *m, float *w,
                armpl_singlecomplex_t *z, const armpl_int_t *ldz,
                armpl_singlecomplex_t *work, float *rwork, armpl_int_t *iwork,
                armpl_int_t *ifail, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found; = 'V': all eigenvalues in the half-open interval (VL,VU] will be found; = 'I': the IL-th through IU-th eigenvalues will be found.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangles of A and B are stored; = 'L': Lower triangles of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

KA Input parameter.

KA is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KA >= 0.

KB Input parameter.

KB is INTEGER

The number of superdiagonals of the matrix B if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KB >= 0.

AB Input and output parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the Hermitian band matrix A, stored in the first ka+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(ka+1+i-j,j) = A(i,j) for max(1,j-ka)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+ka).

On exit, the contents of AB are destroyed.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KA+1.

BB Input and output parameter.

BB is COMPLEX

BB is an array, dimension (LDBB, N). On entry, the upper or lower triangle of the Hermitian band matrix B, stored in the first kb+1 rows of the array. The j-th column of B is stored in the j-th column of the array BB as

follows: if UPLO = 'U', BB(kb+1+i-j,j) = B(i,j) for max(1,j-kb)<=i<=j; if UPLO = 'L', BB(1+i-j,j) = B(i,j) for j<=i<=min(n,j+kb).

On exit, the factor S from the split Cholesky factorization B = S^H *S, as returned by CPBSTF.

LDBB Input parameter.

LDBB is INTEGER

The leading dimension of the array BB. LDBB >= KB+1.

Q Output parameter.

Q is COMPLEX

Q is an array, dimension (LDQ, N). If JOBZ = 'V', the n-by-n matrix used in the reduction of A*x = (lambda)*B*x to standard form, i.e. C*x = (lambda)*x, and consequently C to tridiagonal form. If JOBZ = 'N', the array Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. If JOBZ = 'N', LDQ >= 1. If JOBZ = 'V', LDQ >= max(1, N).

VL Input parameter.

VL is REAL

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is REAL

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is REAL

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing AP to tridiagonal form.

Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*SLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*SLAMCH('S').

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the matrix Z of eigenvectors, with the i-th column of Z holding the eigenvector associated with W(i). The eigenvectors are normalized so that Z^H *B* Z = I. If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= N.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (7*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (5*N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (N)

If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is: <= N: then i eigenvectors failed to converge. Their indices are stored in array IFAIL. > N: if INFO = N + i, for 1 <= i <= N, then CPBSTF returned INFO = i: B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *zhbgvx*. It also exists with a native C interface as *LAPACKE_chbgvx*.

4.12.5 chegst

chegst reduces a complex Hermitian-definite generalized eigenproblem to standard form.

If ITYPE = 1, the problem is A*x = lambda*B*x, and A is overwritten by inv(U^H)*A*inv(U) or inv(L)*A*inv(L^H)

If ITYPE = 2 or 3, the problem is A*B*x = lambda*x or B*A*x = lambda*x, and A is overwritten by U*A*U^H or L^H *A*L.

B must have been previously factorized as U^H *U or L*L^H by CPOTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine chegst(ITYPE, UPLO, N, A, LDA, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void chegst_(const armpl_int_t *itype, const char *uplo, const armpl_int_t *n,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const armpl_singlecomplex_t *b, const armpl_int_t *ldb,
             armpl_int_t *info, ... );
```

Parameters

ITYPE Input parameter.

ITYPE is INTEGER

= 1: compute inv(U^H)*A*inv(U) or inv(L)*A*inv(L^H); = 2 or 3: compute U*A*U^H or L^H *A*L.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored and B is factored as U^H *U; = 'L': Lower triangle of A is stored and B is factored as L*L^H .

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the transformed matrix, stored in the same format as A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, N). The triangular factor from the Cholesky factorization of B, as returned by CPOTRF.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zhegst*. It also exists with a native C interface as *LAPACKE_chegst*.

4.12.6 chegv

chegv computes all the eigenvalues, and optionally, the eigenvectors of a complex generalized Hermitian-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x. Here A and B are assumed to be Hermitian and B is also positive definite.

Syntax

Fortran specification:

```
use armpl_library

subroutine chegv(ITYPE, JOBZ, UPLO, N, A, LDA, B, LDB, W, WORK, LWORK, RWORK,
                 INFO)
```

C specification:

```
#include "armpl.h"

void chegv_(const armpl_int_t *itype, const char *jobz, const char *uplo,
            const armpl_int_t *n, armpl_singlecomplex_t *a,
            const armpl_int_t *lda, armpl_singlecomplex_t *b,
            const armpl_int_t *ldb, float *w, armpl_singlecomplex_t *work,
            const armpl_int_t *lwork, float *rwork, armpl_int_t *info, ... );
```

Parameters

ITYPE Input parameter.

ITYPE is INTEGER

Specifies the problem type to be solved: = 1: A*x = (lambda)*B*x = 2: A*B*x = (lambda)*x = 3: B*A*x = (lambda)*x

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangles of A and B are stored; = 'L': Lower triangles of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A.

On exit, if JOBZ = 'V', then if INFO = 0, A contains the matrix Z of eigenvectors. The eigenvectors are normalized as follows: if ITYPE = 1 or 2, Z^H *B*Z = I; if ITYPE = 3, Z^H *inv(B)*Z = I. If JOBZ = 'N', then on exit the upper triangle (if UPLO='U') or the lower triangle (if UPLO='L') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, N). On entry, the Hermitian positive definite matrix B. If UPLO = 'U', the leading N-by-N upper triangular part of B contains the upper triangular part of the matrix B. If UPLO = 'L', the leading N-by-N lower triangular part of B contains the lower triangular part of the matrix B.

On exit, if INFO <= N, the part of B containing the matrix is overwritten by the triangular factor U or L from the Cholesky factorization B = U^H *U or B = L*L^H .

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= max(1,2*N-1). For optimal efficiency, LWORK >= (NB+1)*N, where NB is the blocksize for CHETRD returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (max(1, 3*N-2)) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: CPOTRF or CHEEV returned an error code: <= N: if INFO = i, CHEEV failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero; > N: if INFO = N + i, for 1 <= i <= N, then the leading minor of

order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *zhegv*. It also exists with a native C interface as *LAPACKE_chegv*.

4.12.7 chegvd

chegvd computes all the eigenvalues, and optionally, the eigenvectors of a complex generalized Hermitian-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x. Here A and B are assumed to be Hermitian and B is also positive definite. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine chegvd(ITYPE, JOBZ, UPLO, N, A, LDA, B, LDB, W, WORK, LWORK, RWORK,
                  LRWORK, IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void chegvd_(const armpl_int_t *itype, const char *jobz, const char *uplo,
             const armpl_int_t *n, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, float *w, armpl_singlecomplex_t *work,
             const armpl_int_t *lwork, float *rwork,
             const armpl_int_t *lrwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

ITYPE Input parameter.

ITYPE is INTEGER

Specifies the problem type to be solved: = 1: A*x = (lambda)*B*x = 2: A*B*x = (lambda)*x = 3: B*A*x = (lambda)*x

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangles of A and B are stored; = 'L': Lower triangles of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A.

On exit, if JOBZ = 'V', then if INFO = 0, A contains the matrix Z of eigenvectors. The eigenvectors are normalized as follows: if ITYPE = 1 or 2, Z^H *B*Z = I; if ITYPE = 3, Z^H *inv(B)*Z = I. If JOBZ = 'N', then on exit the upper triangle (if UPLO='U') or the lower triangle (if UPLO='L') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, N). On entry, the Hermitian matrix B. If UPLO = 'U', the leading N-by-N upper triangular part of B contains the upper triangular part of the matrix B. If UPLO = 'L', the leading N-by-N lower triangular part of B contains the lower triangular part of the matrix B.

On exit, if INFO <= N, the part of B containing the matrix is overwritten by the triangular factor U or L from the Cholesky factorization B = U^H *U or B = L*L^H .

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. If N <= 1, LWORK >= 1. If JOBZ = 'N' and N > 1, LWORK >= N + 1. If JOBZ = 'V' and N > 1, LWORK >= 2* N + N**2.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (MAX(1, LRWORK)). On exit, if INFO = 0, RWORK(1) returns the optimal LRWORK.

LRWORK Input parameter.

LRWORK is INTEGER

The dimension of the array RWORK. If N <= 1, LRWORK >= 1. If JOBZ = 'N' and N > 1, LRWORK >= N. If JOBZ = 'V' and N > 1, LRWORK >= 1 + 5* N + 2*N**2.

If LRWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. If N <= 1, LIWORK >= 1. If JOBZ = 'N' and N > 1, LIWORK >= 1. If JOBZ = 'V' and N > 1, LIWORK >= 3 + 5*N.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: CPOTRF or CHEEVD returned an error code: <= N: if INFO = i and JOBZ = 'N', then the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero; if INFO = i and JOBZ = 'V', then the algorithm failed to compute an eigenvalue while working on the submatrix lying in rows and columns INFO/(N+1) through mod(INFO,N+1); > N: if INFO = N + i, for 1 <= i <= N, then the leading minor of order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *zhegvd*. It also exists with a native C interface as *LAPACKE_chegvd*.

4.12.8 chegvx

chegvx computes selected eigenvalues, and optionally, eigenvectors of a complex generalized Hermitian-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x. Here A and B are assumed to be Hermitian and B is also positive definite. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine chegvx(ITYPE, JOBZ, RANGE, UPLO, N, A, LDA, B, LDB, VL, VU, IL, IU,
                  ABSTOL, M, W, Z, LDZ, WORK, LWORK, RWORK, IWORK, IFAIL,
                  INFO)
```

C specification:

```
#include "armpl.h"

void chegvx_(const armpl_int_t *itype, const char *jobz, const char *range,
             const char *uplo, const armpl_int_t *n, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, const float *vl, const float *vu,
             const armpl_int_t *il, const armpl_int_t *iu,
             const float *abstol, armpl_int_t *m, float *w,
             armpl_singlecomplex_t *z, const armpl_int_t *ldz,
             armpl_singlecomplex_t *work, const armpl_int_t *lwork,
             float *rwork, armpl_int_t *iwork, armpl_int_t *ifail,
             armpl_int_t *info, ... );
```

Parameters

ITYPE Input parameter.

ITYPE is INTEGER

Specifies the problem type to be solved: = 1: A*x = (lambda)*B*x = 2: A*B*x = (lambda)*x = 3: B*A*x = (lambda)*x

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found. = 'V': all eigenvalues in the half-open interval (VL,VU] will be found. = 'I': the IL-th through IU-th eigenvalues will be found.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangles of A and B are stored; = 'L': Lower triangles of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A.

On exit, the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, N). On entry, the Hermitian matrix B. If UPLO = 'U', the leading N-by-N upper triangular part of B contains the upper triangular part of the matrix B. If UPLO = 'L', the leading N-by-N lower triangular part of B contains the lower triangular part of the matrix B.

On exit, if INFO <= N, the part of B containing the matrix is overwritten by the triangular factor U or L from the Cholesky factorization B = U^H *U or B = L*L^H .

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

VL Input parameter.

VL is REAL

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is REAL

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is REAL

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing C to tridiagonal form, where C is the symmetric matrix of the standard symmetric problem to which the generalized problem is transformed.

Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*SLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*SLAMCH('S').

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is REAL

W is an array, dimension (N). The first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'N', then Z is not referenced. If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). The eigenvectors are normalized as follows: if ITYPE = 1 or 2, Z^T *B* Z = I; if ITYPE = 3, Z^T *inv(B)* Z = I.

If an eigenvector fails to converge, then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= max(1,2*N). For optimal efficiency, LWORK >= (NB+1)*N, where NB is the blocksize for CHETRD returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (7*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (5*N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (N)

If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: CPOTRF or CHEEVX returned an error code: <= N: if INFO = i, CHEEVX failed to converge; i eigenvectors failed to converge. Their indices are stored in array IFAIL. > N: if INFO = N + i, for 1 <= i <= N, then the leading minor of order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *zhegvx*. It also exists with a native C interface as *LAPACKE_chegvx*.

4.12.9 chpgst

chpgst reduces a complex Hermitian-definite generalized eigenproblem to standard form, using packed storage.

If ITYPE = 1, the problem is A*x = lambda*B*x, and A is overwritten by inv(U^H)*A*inv(U) or inv(L)*A*inv(L^H)

If ITYPE = 2 or 3, the problem is A*B*x = lambda*x or B*A*x = lambda*x, and A is overwritten by U*A*U^H or L^H *A*L.

B must have been previously factorized as U^H *U or L*L^H by CPPTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine chpgst(ITYPE, UPLO, N, AP, BP, INFO)
```

C specification:

```
#include "armpl.h"

void chpgst_(const armpl_int_t *itype, const char *uplo, const armpl_int_t *n,
             armpl_singlecomplex_t *ap, const armpl_singlecomplex_t *bp,
             armpl_int_t *info, ... );
```

Parameters

ITYPE Input parameter.

ITYPE is INTEGER

= 1: compute inv(U^H)*A*inv(U) or inv(L)*A*inv(L^H); = 2 or 3: compute U*A*U^H or L^H *A*L.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored and B is factored as U^H *U; = 'L': Lower triangle of A is stored and B is factored as L*L^H .

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

AP Input and output parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the Hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

On exit, if INFO = 0, the transformed matrix, stored in the same format as A.

BP Input parameter.

BP is COMPLEX

BP is an array, dimension (N*(N+1)/2). The triangular factor from the Cholesky factorization of B, stored in the same format as A, as returned by CPPTRF.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zhpgst*. It also exists with a native C interface as *LAPACKE_chpgst*.

4.12.10 chpgv

chpgv computes all the eigenvalues and, optionally, the eigenvectors of a complex generalized Hermitian-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x. Here A and B are assumed to be Hermitian, stored in packed format, and B is also positive definite.

Syntax

Fortran specification:

```
use armpl_library

subroutine chpgv(ITYPE, JOBZ, UPLO, N, AP, BP, W, Z, LDZ, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void chpgv_(const armpl_int_t *itype, const char *jobz, const char *uplo,
            const armpl_int_t *n, armpl_singlecomplex_t *ap,
            armpl_singlecomplex_t *bp, float *w, armpl_singlecomplex_t *z,
            const armpl_int_t *ldz, armpl_singlecomplex_t *work, float *rwork,
            armpl_int_t *info, ... );
```

Parameters

ITYPE Input parameter.

ITYPE is INTEGER

Specifies the problem type to be solved: = 1: A*x = (lambda)*B*x = 2: A*B*x = (lambda)*x = 3: B*A*x = (lambda)*x

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangles of A and B are stored; = 'L': Lower triangles of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

AP Input and output parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the Hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n.

On exit, the contents of AP are destroyed.

BP Input and output parameter.

BP is COMPLEX

BP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the Hermitian matrix B, packed columnwise in a linear array. The j-th column of B is stored in the array BP as follows: if UPLO = 'U', BP(i + (j-1)*j/2) = B(i,j) for 1<=i<=j; if UPLO = 'L', BP(i + (j-1)*(2*n-j)/2) = B(i,j) for j<=i<=n.

On exit, the triangular factor U or L from the Cholesky factorization B = U^H *U or B = L*L^H , in the same storage format as B.

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the matrix Z of eigenvectors. The eigenvectors are normalized as follows: if ITYPE = 1 or 2, Z^H *B* Z = I; if ITYPE = 3, Z^H *inv(B)* Z = I. If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (max(1, 2*N-1)) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (max(1, 3*N-2)) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: CPPTRF or CHPEV returned an error code: <= N: if INFO = i, CHPEV failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not convergeto zero; > N: if INFO = N + i, for 1 <= i <= n, then the leading minor of order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *zhpgv*. It also exists with a native C interface as *LAPACKE_chpgv*.

4.12.11 chpgvd

`chpgvd` computes all the eigenvalues and, optionally, the eigenvectors of a complex generalized Hermitian-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x. Here A and B are assumed to be Hermitian, stored in packed format, and B is also positive definite. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine chpgvd(ITYPE, JOBZ, UPLO, N, AP, BP, W, Z, LDZ, WORK, LWORK, RWORK,
                  LRWORK, IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void chpgvd_(const armpl_int_t *itype, const char *jobz, const char *uplo,
             const armpl_int_t *n, armpl_singlecomplex_t *ap,
             armpl_singlecomplex_t *bp, float *w, armpl_singlecomplex_t *z,
             const armpl_int_t *ldz, armpl_singlecomplex_t *work,
             const armpl_int_t *lwork, float *rwork,
             const armpl_int_t *lrwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

ITYPE Input parameter.

ITYPE is INTEGER

Specifies the problem type to be solved: = 1: A*x = (lambda)*B*x = 2: A*B*x = (lambda)*x = 3: B*A*x = (lambda)*x

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangles of A and B are stored; = 'L': Lower triangles of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

AP Input and output parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the Hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n.

On exit, the contents of AP are destroyed.

BP Input and output parameter.

BP is COMPLEX

BP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the Hermitian matrix B, packed columnwise in a linear array. The j-th column of B is stored in the array BP as follows: if UPLO = 'U', BP(i + (j-1)*j/2) = B(i,j) for 1<=i<=j; if UPLO = 'L', BP(i + (j-1)*(2*n-j)/2) = B(i,j) for j<=i<=n.

On exit, the triangular factor U or L from the Cholesky factorization B = U^H *U or B = L*L^H , in the same storage format as B.

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the matrix Z of eigenvectors. The eigenvectors are normalized as follows: if ITYPE = 1 or 2, Z^H *B* Z = I; if ITYPE = 3, Z^H *inv(B)* Z = I. If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the required LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of array WORK. If N <= 1, LWORK >= 1. If JOBZ = 'N' and N > 1, LWORK >= N. If JOBZ = 'V' and N > 1, LWORK >= 2*N.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the required sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (MAX(1, LRWORK)). On exit, if INFO = 0, RWORK(1) returns the required LRWORK.

LRWORK Input parameter.

LRWORK is INTEGER

The dimension of array RWORK. If N <= 1, LRWORK >= 1. If JOBZ = 'N' and N > 1, LRWORK >= N. If JOBZ = 'V' and N > 1, LRWORK >= 1 + 5* N + 2*N**2.

If LRWORK = -1, then a workspace query is assumed; the routine only calculates the required sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the required LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of array IWORK. If JOBZ = 'N' or N <= 1, LIWORK >= 1. If JOBZ = 'V' and N > 1, LIWORK >= 3 + 5*N.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the required sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: CPPTRF or CHPEVD returned an error code: <= N: if INFO = i, CHPEVD failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not convergeto zero; > N: if INFO = N + i, for 1 <= i <= n, then the leading minor of order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *zhpgvd*. It also exists with a native C interface as *LAPACKE_chpgvd*.

4.12.12 chpgvx

chpgvx computes selected eigenvalues and, optionally, eigenvectors of a complex generalized Hermitian-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x. Here A and B are assumed to be Hermitian, stored in packed format, and B is also positive definite. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine chpgvx(ITYPE, JOBZ, RANGE, UPLO, N, AP, BP, VL, VU, IL, IU, ABSTOL,
                  M, W, Z, LDZ, WORK, RWORK, IWORK, IFAIL, INFO)
```

C specification:

```
#include "armpl.h"

void chpgvx_(const armpl_int_t *itype, const char *jobz, const char *range,
             const char *uplo, const armpl_int_t *n,
             armpl_singlecomplex_t *ap, armpl_singlecomplex_t *bp,
             const float *vl, const float *vu, const armpl_int_t *il,
             const armpl_int_t *iu, const float *abstol, armpl_int_t *m,
             float *w, armpl_singlecomplex_t *z, const armpl_int_t *ldz,
             armpl_singlecomplex_t *work, float *rwork, armpl_int_t *iwork,
             armpl_int_t *ifail, armpl_int_t *info, ... );
```

Parameters

ITYPE Input parameter.

ITYPE is INTEGER

Specifies the problem type to be solved: = 1: A*x = (lambda)*B*x = 2: A*B*x = (lambda)*x = 3: B*A*x = (lambda)*x

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found; = 'V': all eigenvalues in the half-open interval (VL,VU] will be found; = 'I': the IL-th through IU-th eigenvalues will be found.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangles of A and B are stored; = 'L': Lower triangles of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

AP Input and output parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the Hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n.

On exit, the contents of AP are destroyed.

BP Input and output parameter.

BP is COMPLEX

BP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the Hermitian matrix B, packed columnwise in a linear array. The j-th column of B is stored in the array BP as follows: if UPLO = 'U', BP(i + (j-1)*j/2) = B(i,j) for 1<=i<=j; if UPLO = 'L', BP(i + (j-1)*(2*n-j)/2) = B(i,j) for j<=i<=n.

On exit, the triangular factor U or L from the Cholesky factorization B = U^H *U or B = L*L^H , in the same storage format as B.

VL Input parameter.

VL is REAL

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is REAL

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is REAL

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing AP to tridiagonal form.

Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*SLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*SLAMCH('S').

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is REAL

W is an array, dimension (N). On normal exit, the first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, N). If JOBZ = 'N', then Z is not referenced. If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). The eigenvectors are normalized as follows: if ITYPE = 1 or 2, Z^H *B* Z = I; if ITYPE = 3, Z^H *inv(B)* Z = I.

If an eigenvector fails to converge, then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (7*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (5*N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (N)

If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: CPPTRF or CHPEVX returned an error code: <= N: if INFO = i, CHPEVX failed to converge; i eigenvectors failed to converge. Their indices are stored in array IFAIL. > N: if INFO = N + i, for 1 <= i <= n, then the leading minor of order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *zhpgvx*. It also exists with a native C interface as *LAPACKE_chpgvx*.

4.12.13 cpbstf

cpbstf computes a split Cholesky factorization of a complex Hermitian positive definite band matrix A.

This routine is designed to be used in conjunction with CHBGST.

The factorization has the form A = S^H *S where S is a band matrix of the same bandwidth as A and the following structure:

```
S = ( U    )
    ( M  L )
```

where U is upper triangular of order m = (n+kd)/2, and L is lower triangular of order n-m.

Syntax

Fortran specification:

```
use armpl_library

subroutine cpbstf(UPLO, N, KD, AB, LDAB, INFO)
```

C specification:

```
#include "armpl.h"

void cpbstf_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
             armpl_singlecomplex_t *ab, const armpl_int_t *ldab,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the Hermitian band matrix A, stored in the first kd+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, if INFO = 0, the factor S from the split Cholesky factorization A = S^H *S. See Further Details.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the factorization could not be completed, because the updated element a(i,i) was negative; the matrix A is not positive definite.

Related Information

For this routine in other precisions, please see *dpbstf*, *spbstf* and *zpbstf*. It also exists with a native C interface as *LAPACKE_cpbstf*.

4.12.14 dpbstf

dpbstf computes a split Cholesky factorization of a real symmetric positive definite band matrix A.

This routine is designed to be used in conjunction with DSBGST.

The factorization has the form A = S^T *S where S is a band matrix of the same bandwidth as A and the following structure:

```
S = ( U    )
    ( M  L )
```

where U is upper triangular of order m = (n+kd)/2, and L is lower triangular of order n-m.

Syntax

Fortran specification:

```
use armpl_library

subroutine dpbstf(UPLO, N, KD, AB, LDAB, INFO)
```

C specification:

```
#include "armpl.h"

void dpbstf_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
             double *ab, const armpl_int_t *ldab, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first kd+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, if INFO = 0, the factor S from the split Cholesky factorization A = S^T *S. See Further Details.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the factorization could not be completed, because the updated element a(i,i) was negative; the matrix A is not positive definite.

Related Information

For this routine in other precisions, please see *cpbstf*, *spbstf* and *zpbstf*. It also exists with a native C interface as *LAPACKE_dpbstf*.

4.12.15 dsbgst

dsbgst reduces a real symmetric-definite banded generalized eigenproblem A*x = lambda*B*x to standard form C*y = lambda*y, such that C has the same bandwidth as A.

B must have been previously factorized as S^T *S by DPBSTF, using a split Cholesky factorization. A is overwritten by C = X^T *A*X, where X = S**(-1)*Q and Q is an orthogonal matrix chosen to preserve the bandwidth of A.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsbgst(VECT, UPLO, N, KA, KB, AB, LDAB, BB, LDBB, X, LDX, WORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dsbgst_(const char *vect, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *ka, const armpl_int_t *kb, double *ab,
             const armpl_int_t *ldab, const double *bb,
             const armpl_int_t *ldbb, double *x, const armpl_int_t *ldx,
             double *work, armpl_int_t *info, ... );
```

Parameters

VECT Input parameter.

VECT is CHARACTER*1

= 'N': do not form the transformation matrix X; = 'V': form X.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

KA Input parameter.

KA is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KA >= 0.

KB Input parameter.

KB is INTEGER

The number of superdiagonals of the matrix B if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KA >= KB >= 0.

AB Input and output parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first ka+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(ka+1+i-j,j) = A(i,j) for max(1,j-ka)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+ka).

On exit, the transformed matrix X^T *A*X, stored in the same format as A.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KA+1.

BB Input parameter.

BB is DOUBLE PRECISION

BB is an array, dimension (LDBB, N). The banded factor S from the split Cholesky factorization of B, as returned by DPBSTF, stored in the first KB+1 rows of the array.

LDBB Input parameter.

LDBB is INTEGER

The leading dimension of the array BB. LDBB >= KB+1.

X Output parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, N). If VECT = 'V', the n-by-n matrix X. If VECT = 'N', the array X is not referenced.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N) if VECT = 'V'; LDX >= 1 otherwise.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *ssbgst*. It also exists with a native C interface as *LAPACKE_dsbgst*.

4.12.16 dsbgv

dsbgv computes all the eigenvalues, and optionally, the eigenvectors of a real generalized symmetric-definite banded eigenproblem, of the form A*x=(lambda)*B*x. Here A and B are assumed to be symmetric and banded, and B is also positive definite.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsbgv(JOBZ, UPLO, N, KA, KB, AB, LDAB, BB, LDBB, W, Z, LDZ, WORK,
                 INFO)
```

C specification:

```
#include "armpl.h"

void dsbgv_(const char *jobz, const char *uplo, const armpl_int_t *n,
            const armpl_int_t *ka, const armpl_int_t *kb, double *ab,
            const armpl_int_t *ldab, double *bb, const armpl_int_t *ldbb,
```

(continues on next page)

(continued from previous page)

```
            double *w, double *z, const armpl_int_t *ldz, double *work,
            armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangles of A and B are stored; = 'L': Lower triangles of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

KA Input parameter.

KA is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KA >= 0.

KB Input parameter.

KB is INTEGER

The number of superdiagonals of the matrix B if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KB >= 0.

AB Input and output parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first ka+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(ka+1+i-j,j) = A(i,j) for max(1,j-ka)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+ka).

On exit, the contents of AB are destroyed.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KA+1.

BB Input and output parameter.

BB is DOUBLE PRECISION

BB is an array, dimension (LDBB, N). On entry, the upper or lower triangle of the symmetric band matrix B, stored in the first kb+1 rows of the array. The j-th column of B is stored in the j-th column of the array BB as follows: if UPLO = 'U', BB(kb+1+i-j,j) = B(i,j) for max(1,j-kb)<=i<=j; if UPLO = 'L', BB(1+i-j,j) = B(i,j) for j<=i<=min(n,j+kb).

On exit, the factor S from the split Cholesky factorization B = S^T *S, as returned by DPBSTF.

LDBB Input parameter.

LDBB is INTEGER

The leading dimension of the array BB. LDBB >= KB+1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the matrix Z of eigenvectors, with the i-th column of Z holding the eigenvector associated with W(i). The eigenvectors are normalized so that Z^T *B* Z = I. If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= N.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (3*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is: <= N: the algorithm failed to converge: i off-diagonal elements of an intermediate tridiagonal form did not converge to zero; > N: if INFO = N + i, for 1 <= i <= N, then DPBSTF returned INFO = i: B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *ssbgv*. It also exists with a native C interface as *LAPACKE_dsbgv*.

4.12.17 dsbgvd

dsbgvd computes all the eigenvalues, and optionally, the eigenvectors of a real generalized symmetric-definite banded eigenproblem, of the form A*x=(lambda)*B*x. Here A and B are assumed to be symmetric and banded, and B is also positive definite. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsbgvd(JOBZ, UPLO, N, KA, KB, AB, LDAB, BB, LDBB, W, Z, LDZ, WORK,
                  LWORK, IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dsbgvd_(const char *jobz, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *ka, const armpl_int_t *kb, double *ab,
             const armpl_int_t *ldab, double *bb, const armpl_int_t *ldbb,
             double *w, double *z, const armpl_int_t *ldz, double *work,
             const armpl_int_t *lwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangles of A and B are stored; = 'L': Lower triangles of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

KA Input parameter.

KA is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KA >= 0.

KB Input parameter.

KB is INTEGER

The number of superdiagonals of the matrix B if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KB >= 0.

AB Input and output parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first ka+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(ka+1+i-j,j) = A(i,j) for max(1,j-ka)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+ka).

On exit, the contents of AB are destroyed.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KA+1.

BB Input and output parameter.

BB is DOUBLE PRECISION

BB is an array, dimension (LDBB, N). On entry, the upper or lower triangle of the symmetric band matrix B, stored in the first kb+1 rows of the array. The j-th column of B is stored in the j-th column of the array BB as follows: if UPLO = 'U', BB(ka+1+i-j,j) = B(i,j) for max(1,j-kb)<=i<=j; if UPLO = 'L', BB(1+i-j,j) = B(i,j) for j<=i<=min(n,j+kb).

On exit, the factor S from the split Cholesky factorization B = S^T *S, as returned by DPBSTF.

LDBB Input parameter.

LDBB is INTEGER

The leading dimension of the array BB. LDBB >= KB+1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the matrix Z of eigenvectors, with the i-th column of Z holding the eigenvector associated with W(i). The eigenvectors are normalized so Z^T *B* Z = I. If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If N <= 1, LWORK >= 1. If JOBZ = 'N' and N > 1, LWORK >= 2*N. If JOBZ = 'V' and N > 1, LWORK >= 1 + 5* N + 2*N**2.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if LIWORK > 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. If JOBZ = 'N' or N <= 1, LIWORK >= 1. If JOBZ = 'V' and N > 1, LIWORK >= 3 + 5*N.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is: <= N: the algorithm failed to converge: i off-diagonal elements of an intermediate tridiagonal form did not converge to zero; > N: if INFO = N + i, for 1 <= i <= N, then DPBSTF returned INFO = i: B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *ssbgvd*. It also exists with a native C interface as *LAPACKE_dsbgvd*.

4.12.18 dsbgvx

`dsbgvx` computes selected eigenvalues, and optionally, eigenvectors of a real generalized symmetric-definite banded eigenproblem, of the form A*x=(lambda)*B*x. Here A and B are assumed to be symmetric and banded, and B is also positive definite. Eigenvalues and eigenvectors can be selected by specifying either all eigenvalues, a range of values or a range of indices for the desired eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsbgvx(JOBZ, RANGE, UPLO, N, KA, KB, AB, LDAB, BB, LDBB, Q, LDQ,
                  VL, VU, IL, IU, ABSTOL, M, W, Z, LDZ, WORK, IWORK, IFAIL,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dsbgvx_(const char *jobz, const char *range, const char *uplo,
             const armpl_int_t *n, const armpl_int_t *ka,
             const armpl_int_t *kb, double *ab, const armpl_int_t *ldab,
             double *bb, const armpl_int_t *ldbb, double *q,
             const armpl_int_t *ldq, const double *vl, const double *vu,
             const armpl_int_t *il, const armpl_int_t *iu,
             const double *abstol, armpl_int_t *m, double *w, double *z,
             const armpl_int_t *ldz, double *work, armpl_int_t *iwork,
             armpl_int_t *ifail, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

`JOBZ` is `CHARACTER*1`

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

RANGE Input parameter.

`RANGE` is `CHARACTER*1`

= 'A': all eigenvalues will be found. = 'V': all eigenvalues in the half-open interval (`VL`,VU] will be found. = 'I': the IL-th through IU-th eigenvalues will be found.

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

= 'U': Upper triangles of A and B are stored; = 'L': Lower triangles of A and B are stored.

N Input parameter.

`N` is `INTEGER`

The order of the matrices A and B. `N` >= 0.

KA Input parameter.

KA is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KA >= 0.

KB Input parameter.

KB is INTEGER

The number of superdiagonals of the matrix B if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KB >= 0.

AB Input and output parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first ka+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(ka+1+i-j,j) = A(i,j) for max(1,j-ka)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+ka).

On exit, the contents of AB are destroyed.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KA+1.

BB Input and output parameter.

BB is DOUBLE PRECISION

BB is an array, dimension (LDBB, N). On entry, the upper or lower triangle of the symmetric band matrix B, stored in the first kb+1 rows of the array. The j-th column of B is stored in the j-th column of the array BB as follows: if UPLO = 'U', BB(ka+1+i-j,j) = B(i,j) for max(1,j-kb)<=i<=j; if UPLO = 'L', BB(1+i-j,j) = B(i,j) for j<=i<=min(n,j+kb).

On exit, the factor S from the split Cholesky factorization B = S^T *S, as returned by DPBSTF.

LDBB Input parameter.

LDBB is INTEGER

The leading dimension of the array BB. LDBB >= KB+1.

Q Output parameter.

Q is DOUBLE PRECISION

Q is an array, dimension (LDQ, N). If JOBZ = 'V', the n-by-n matrix used in the reduction of A*x = (lambda)*B*x to standard form, i.e. C*x = (lambda)*x, and consequently C to tridiagonal form. If JOBZ = 'N', the array Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. If JOBZ = 'N', LDQ >= 1. If JOBZ = 'V', LDQ >= max(1, N).

VL Input parameter.

VL is DOUBLE PRECISION

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is DOUBLE PRECISION

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is DOUBLE PRECISION

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing A to tridiagonal form.

Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*DLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*DLAMCH('S').

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the matrix Z of eigenvectors, with the i-th column of Z holding the eigenvector associated with W(i). The eigenvectors are normalized so Z^T *B* Z = I. If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (7*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (5*N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (M)

If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvalues that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0 : successful exit < 0 : if INFO = -i, the i-th argument had an illegal value <= N: if INFO = i, then i eigenvectors failed to converge. Their indices are stored in IFAIL. > N : DPBSTF returned an error code; i.e., if INFO = N + i, for 1 <= i <= N, then the leading minor of order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *ssbgvx*. It also exists with a native C interface as *LAPACKE_dsbgvx*.

4.12.19 dspgst

dspgst reduces a real symmetric-definite generalized eigenproblem to standard form, using packed storage.

If ITYPE = 1, the problem is A*x = lambda*B*x, and A is overwritten by inv(U^T)*A*inv(U) or inv(L)*A*inv(L^T)

If ITYPE = 2 or 3, the problem is A*B*x = lambda*x or B*A*x = lambda*x, and A is overwritten by U*A*U^T or L^T*A*L.

B must have been previously factorized as U^T*U or L*L^T by DPPTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine dspgst(ITYPE, UPLO, N, AP, BP, INFO)
```

C specification:

```
#include "armpl.h"

void dspgst_(const armpl_int_t *itype, const char *uplo, const armpl_int_t *n,
             double *ap, const double *bp, armpl_int_t *info, ... );
```

Parameters

ITYPE Input parameter.

ITYPE is INTEGER

= 1: compute inv(U^T)*A*inv(U) or inv(L)*A*inv(L^T); = 2 or 3: compute U*A*U^T or L^T*A*L.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored and B is factored as U^T*U; = 'L': Lower triangle of A is stored and B is factored as L*L^T .

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

AP Input and output parameter.

AP is DOUBLE PRECISION

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

On exit, if INFO = 0, the transformed matrix, stored in the same format as A.

BP Input parameter.

BP is DOUBLE PRECISION

BP is an array, dimension (N*(N+1)/2). The triangular factor from the Cholesky factorization of B, stored in the same format as A, as returned by DPPTRF.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *sspgst*. It also exists with a native C interface as *LAPACKE_dspgst*.

4.12.20 dspgv

dspgv computes all the eigenvalues and, optionally, the eigenvectors of a real generalized symmetric-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x. Here A and B are assumed to be symmetric, stored in packed format, and B is also positive definite.

Syntax

Fortran specification:

```
use armpl_library

subroutine dspgv(ITYPE, JOBZ, UPLO, N, AP, BP, W, Z, LDZ, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dspgv_(const armpl_int_t *itype, const char *jobz, const char *uplo,
            const armpl_int_t *n, double *ap, double *bp, double *w,
            double *z, const armpl_int_t *ldz, double *work,
            armpl_int_t *info, ... );
```

Parameters

ITYPE Input parameter.

ITYPE is INTEGER

Specifies the problem type to be solved: = 1: A*x = (lambda)*B*x = 2: A*B*x = (lambda)*x = 3: B*A*x = (lambda)*x

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangles of A and B are stored; = 'L': Lower triangles of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

AP Input and output parameter.

AP is DOUBLE PRECISION

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n.

On exit, the contents of AP are destroyed.

BP Input and output parameter.

BP is DOUBLE PRECISION

BP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix B, packed columnwise in a linear array. The j-th column of B is stored in the array BP as follows: if UPLO = 'U', BP(i + (j-1)*j/2) = B(i,j) for 1<=i<=j; if UPLO = 'L', BP(i + (j-1)*(2*n-j)/2) = B(i,j) for j<=i<=n.

On exit, the triangular factor U or L from the Cholesky factorization B = U^T *U or B = L*L^T , in the same storage format as B.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the matrix Z of eigenvectors. The eigenvectors are normalized as follows: if ITYPE = 1 or 2, Z^T *B* Z = I; if ITYPE = 3, Z^T *inv(B)* Z = I. If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (3*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: DPPTRF or DSPEV returned an error code: <= N: if INFO = i, DSPEV failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero. > N: if INFO = n + i, for 1 <= i <= n, then the leading minor of order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *sspgv*. It also exists with a native C interface as *LAPACKE_dspgv*.

4.12.21 dspgvd

`dspgvd` computes all the eigenvalues, and optionally, the eigenvectors of a real generalized symmetric-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x. Here A and B are assumed to be symmetric, stored in packed format, and B is also positive definite. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine dspgvd(ITYPE, JOBZ, UPLO, N, AP, BP, W, Z, LDZ, WORK, LWORK, IWORK,
                  LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dspgvd_(const armpl_int_t *itype, const char *jobz, const char *uplo,
             const armpl_int_t *n, double *ap, double *bp, double *w,
             double *z, const armpl_int_t *ldz, double *work,
             const armpl_int_t *lwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

ITYPE Input parameter.

`ITYPE` is `INTEGER`

Specifies the problem type to be solved: = 1: A*x = (lambda)*B*x = 2: A*B*x = (lambda)*x = 3: B*A*x = (lambda)*x

JOBZ Input parameter.

`JOBZ` is `CHARACTER*1`

= ‘`N`’: Compute eigenvalues only; = ‘V’: Compute eigenvalues and eigenvectors.

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

= ‘`U`’: Upper triangles of A and B are stored; = ‘`L`’: Lower triangles of A and B are stored.

N Input parameter.

`N` is `INTEGER`

The order of the matrices A and B. `N` >= 0.

AP Input and output parameter.

AP is DOUBLE PRECISION

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n.

On exit, the contents of AP are destroyed.

BP Input and output parameter.

BP is DOUBLE PRECISION

BP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix B, packed columnwise in a linear array. The j-th column of B is stored in the array BP as follows: if UPLO = 'U', BP(i + (j-1)*j/2) = B(i,j) for 1<=i<=j; if UPLO = 'L', BP(i + (j-1)*(2*n-j)/2) = B(i,j) for j<=i<=n.

On exit, the triangular factor U or L from the Cholesky factorization B = U^T *U or B = L*L^T , in the same storage format as B.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the matrix Z of eigenvectors. The eigenvectors are normalized as follows: if ITYPE = 1 or 2, Z^T *B* Z = I; if ITYPE = 3, Z^T *inv(B)* Z = I. If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the required LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If N <= 1, LWORK >= 1. If JOBZ = 'N' and N > 1, LWORK >= 2*N. If JOBZ = 'V' and N > 1, LWORK >= 1 + 6* N + 2*N**2.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the required sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the required LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. If JOBZ = 'N' or N <= 1, LIWORK >= 1. If JOBZ = 'V' and N > 1, LIWORK >= 3 + 5*N.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the required sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: DPPTRF or DSPEVD returned an error code: <= N: if INFO = i, DSPEVD failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero; > N: if INFO = N + i, for 1 <= i <= N, then the leading minor of order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *sspgvd*. It also exists with a native C interface as *LAPACKE_dspgvd*.

4.12.22 dspgvx

dspgvx computes selected eigenvalues, and optionally, eigenvectors of a real generalized symmetric-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x. Here A and B are assumed to be symmetric, stored in packed storage, and B is also positive definite. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine dspgvx(ITYPE, JOBZ, RANGE, UPLO, N, AP, BP, VL, VU, IL, IU, ABSTOL,
                  M, W, Z, LDZ, WORK, IWORK, IFAIL, INFO)
```

C specification:

```
#include "armpl.h"

void dspgvx_(const armpl_int_t *itype, const char *jobz, const char *range,
             const char *uplo, const armpl_int_t *n, double *ap, double *bp,
             const double *vl, const double *vu, const armpl_int_t *il,
             const armpl_int_t *iu, const double *abstol, armpl_int_t *m,
             double *w, double *z, const armpl_int_t *ldz, double *work,
             armpl_int_t *iwork, armpl_int_t *ifail, armpl_int_t *info,
             ... );
```

Parameters

ITYPE Input parameter.

ITYPE is INTEGER

Specifies the problem type to be solved: = 1: A*x = (lambda)*B*x = 2: A*B*x = (lambda)*x = 3: B*A*x = (lambda)*x

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found. = 'V': all eigenvalues in the half-open interval (VL,VU] will be found. = 'I': the IL-th through IU-th eigenvalues will be found.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A and B are stored; = 'L': Lower triangle of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrix pencil (A,B). N >= 0.

AP Input and output parameter.

AP is DOUBLE PRECISION

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n.

On exit, the contents of AP are destroyed.

BP Input and output parameter.

BP is DOUBLE PRECISION

BP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix B, packed columnwise in a linear array. The j-th column of B is stored in the array BP as follows: if UPLO = 'U', BP(i + (j-1)*j/2) = B(i,j) for 1<=i<=j; if UPLO = 'L', BP(i + (j-1)*(2*n-j)/2) = B(i,j) for j<=i<=n.

On exit, the triangular factor U or L from the Cholesky factorization $B = U^T *U$ or $B = L*L^T$, in the same storage format as B.

VL Input parameter.

VL is DOUBLE PRECISION

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is DOUBLE PRECISION

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is DOUBLE PRECISION

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing A to tridiagonal form.

Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*DLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*DLAMCH('S').

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). On normal exit, the first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'N', then Z is not referenced. If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). The eigenvectors are normalized as follows: if ITYPE = 1 or 2, Z^T *B* Z = I; if ITYPE = 3, Z^T *inv(B)* Z = I.

If an eigenvector fails to converge, then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (8*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (5*N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (N)

If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: DPPTRF or DSPEVX returned an error code: <= N: if INFO = i, DSPEVX failed to converge; i eigenvectors failed to converge. Their indices are stored in array IFAIL. > N: if INFO = N + i, for 1 <= i <= N, then the leading minor of order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *sspgvx*. It also exists with a native C interface as *LAPACKE_dspgvx*.

4.12.23 dsygst

`dsygst` reduces a real symmetric-definite generalized eigenproblem to standard form.

If ITYPE = 1, the problem is A*x = lambda*B*x, and A is overwritten by inv(U^T)*A*inv(U) or inv(L)*A*inv(L^T)

If ITYPE = 2 or 3, the problem is A*B*x = lambda*x or B*A*x = lambda*x, and A is overwritten by U*A*U^T or L^T*A*L.

B must have been previously factorized as U^T*U or L*L^T by DPOTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsygst(ITYPE, UPLO, N, A, LDA, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void dsygst_(const armpl_int_t *itype, const char *uplo, const armpl_int_t *n,
             double *a, const armpl_int_t *lda, const double *b,
             const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

ITYPE Input parameter.

`ITYPE` is `INTEGER`

= 1: compute inv(U^T)*A*inv(U) or inv(L)*A*inv(L^T); = 2 or 3: compute U*A*U^T or L^T*A*L.

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

= '`U`': Upper triangle of `A` is stored and `B` is factored as U^T*U; = '`L`': Lower triangle of `A` is stored and `B` is factored as L*L^T.

N Input parameter.

`N` is `INTEGER`

The order of the matrices `A` and `B`. `N` >= 0.

A Input and output parameter.

`A` is `DOUBLE PRECISION`

`A` is an array, dimension (`LDA`, `N`). On entry, the symmetric matrix `A`. If `UPLO` = '`U`', the leading N-by-N upper triangular part of `A` contains the upper triangular part of the matrix `A`, and the strictly lower triangular part of `A` is not referenced. If `UPLO` = '`L`', the leading N-by-N lower triangular part of `A` contains the lower triangular part of the matrix `A`, and the strictly upper triangular part of `A` is not referenced.

On exit, if `INFO` = 0, the transformed matrix, stored in the same format as `A`.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). The triangular factor from the Cholesky factorization of B, as returned by DPOTRF.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ssygst*. It also exists with a native C interface as *LAPACKE_dsygst*.

4.12.24 dsygv

dsygv computes all the eigenvalues, and optionally, the eigenvectors of a real generalized symmetric-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x. Here A and B are assumed to be symmetric and B is also positive definite.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsygv(ITYPE, JOBZ, UPLO, N, A, LDA, B, LDB, W, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dsygv_(const armpl_int_t *itype, const char *jobz, const char *uplo,
            const armpl_int_t *n, double *a, const armpl_int_t *lda,
            double *b, const armpl_int_t *ldb, double *w, double *work,
            const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

ITYPE Input parameter.

ITYPE is INTEGER

Specifies the problem type to be solved: = 1: A*x = (lambda)*B*x = 2: A*B*x = (lambda)*x = 3: B*A*x = (lambda)*x

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangles of A and B are stored; = 'L': Lower triangles of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A.

On exit, if JOBZ = 'V', then if INFO = 0, A contains the matrix Z of eigenvectors. The eigenvectors are normalized as follows: if ITYPE = 1 or 2, Z^T *B*Z = I; if ITYPE = 3, Z^T *inv(B)*Z = I. If JOBZ = 'N', then on exit the upper triangle (if UPLO='U') or the lower triangle (if UPLO='L') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). On entry, the symmetric positive definite matrix B. If UPLO = 'U', the leading N-by-N upper triangular part of B contains the upper triangular part of the matrix B. If UPLO = 'L', the leading N-by-N lower triangular part of B contains the lower triangular part of the matrix B.

On exit, if INFO <= N, the part of B containing the matrix is overwritten by the triangular factor U or L from the Cholesky factorization B = U^T *U or B = L*L^T .

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= max(1,3*N-1). For optimal efficiency, LWORK >= (NB+2)*N, where NB is the blocksize for DSYTRD returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: DPOTRF or DSYEV returned an error code: <= N: if INFO = i, DSYEV failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero; > N: if INFO = N + i, for 1 <= i <= N, then the leading minor of order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *ssygv*. It also exists with a native C interface as *LAPACKE_dsygv*.

4.12.25 dsygvd

dsygvd computes all the eigenvalues, and optionally, the eigenvectors of a real generalized symmetric-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x. Here A and B are assumed to be symmetric and B is also positive definite. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsygvd(ITYPE, JOBZ, UPLO, N, A, LDA, B, LDB, W, WORK, LWORK, IWORK,
                  LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dsygvd_(const armpl_int_t *itype, const char *jobz, const char *uplo,
             const armpl_int_t *n, double *a, const armpl_int_t *lda,
             double *b, const armpl_int_t *ldb, double *w, double *work,
             const armpl_int_t *lwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

ITYPE Input parameter.

ITYPE is INTEGER

Specifies the problem type to be solved: = 1: A*x = (lambda)*B*x = 2: A*B*x = (lambda)*x = 3: B*A*x = (lambda)*x

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangles of A and B are stored; = 'L': Lower triangles of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A.

On exit, if JOBZ = 'V', then if INFO = 0, A contains the matrix Z of eigenvectors. The eigenvectors are normalized as follows: if ITYPE = 1 or 2, Z^T *B*Z = I; if ITYPE = 3, Z^T *inv(B)*Z = I. If JOBZ = 'N', then on exit the upper triangle (if UPLO='U') or the lower triangle (if UPLO='L') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). On entry, the symmetric matrix B. If UPLO = 'U', the leading N-by-N upper triangular part of B contains the upper triangular part of the matrix B. If UPLO = 'L', the leading N-by-N lower triangular part of B contains the lower triangular part of the matrix B.

On exit, if INFO <= N, the part of B containing the matrix is overwritten by the triangular factor U or L from the Cholesky factorization B = U^T *U or B = L*L^T .

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If N <= 1, LWORK >= 1. If JOBZ = 'N' and N > 1, LWORK >= 2*N+1. If JOBZ = 'V' and N > 1, LWORK >= 1 + 6* N + 2*N**2.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. If N <= 1, LIWORK >= 1. If JOBZ = 'N' and N > 1, LIWORK >= 1. If JOBZ = 'V' and N > 1, LIWORK >= 3 + 5*N.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: DPOTRF or DSYEVD returned an error code: <= N: if INFO = i and JOBZ = 'N', then the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero; if INFO = i and JOBZ = 'V', then the algorithm failed to compute an eigenvalue while working on the submatrix lying in rows and columns INFO/(N+1) through mod(INFO,N+1); > N: if INFO = N + i, for 1 <= i <= N, then the leading minor of order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *ssygvd*. It also exists with a native C interface as *LAPACKE_dsygvd*.

4.12.26 dsygvx

dsygvx computes selected eigenvalues, and optionally, eigenvectors of a real generalized symmetric-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x. Here A and B are assumed to be symmetric and B is also positive definite. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsygvx(ITYPE, JOBZ, RANGE, UPLO, N, A, LDA, B, LDB, VL, VU, IL, IU,
                  ABSTOL, M, W, Z, LDZ, WORK, LWORK, IWORK, IFAIL, INFO)
```

C specification:

```
#include "armpl.h"

void dsygvx_(const armpl_int_t *itype, const char *jobz, const char *range,
             const char *uplo, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, double *b, const armpl_int_t *ldb,
             const double *vl, const double *vu, const armpl_int_t *il,
```

(continues on next page)

(continued from previous page)

```
                const armpl_int_t *iu, const double *abstol, armpl_int_t *m,
                double *w, double *z, const armpl_int_t *ldz, double *work,
                const armpl_int_t *lwork, armpl_int_t *iwork, armpl_int_t *ifail,
                armpl_int_t *info, ... );
```

Parameters

ITYPE Input parameter.

ITYPE is INTEGER

Specifies the problem type to be solved: = 1: A*x = (lambda)*B*x = 2: A*B*x = (lambda)*x = 3: B*A*x = (lambda)*x

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found. = 'V': all eigenvalues in the half-open interval (VL,VU] will be found. = 'I': the IL-th through IU-th eigenvalues will be found.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A and B are stored; = 'L': Lower triangle of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrix pencil (A, B). N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A.

On exit, the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). On entry, the symmetric matrix B. If UPLO = 'U', the leading N-by-N upper triangular part of B contains the upper triangular part of the matrix B. If UPLO = 'L', the leading N-by-N lower triangular part of B contains the lower triangular part of the matrix B.

On exit, if INFO <= N, the part of B containing the matrix is overwritten by the triangular factor U or L from the Cholesky factorization B = U^T *U or B = L*L^T .

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

VL Input parameter.

VL is DOUBLE PRECISION

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is DOUBLE PRECISION

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is DOUBLE PRECISION

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing C to tridiagonal form, where C is the symmetric matrix of the standard symmetric problem to which the generalized problem is transformed.

Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*DLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*DLAMCH('S').

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). On normal exit, the first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'N', then Z is not referenced. If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). The eigenvectors are normalized as follows: if ITYPE = 1 or 2, Z^T *B* Z = I; if ITYPE = 3, Z^T *inv(B)* Z = I.

If an eigenvector fails to converge, then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= max(1,8*N). For optimal efficiency, LWORK >= (NB+3)*N, where NB is the blocksize for DSYTRD returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (5*N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (N)

If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: DPOTRF or DSYEVX returned an error code: <= N: if INFO = i, DSYEVX failed to converge; i eigenvectors failed to converge. Their indices are stored in array IFAIL. > N: if INFO = N + i, for 1 <= i <= N, then the leading minor of order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *ssygvx*. It also exists with a native C interface as *LAPACKE_dsygvx*.

4.12.27 spbstf

spbstf computes a split Cholesky factorization of a real symmetric positive definite band matrix A.

This routine is designed to be used in conjunction with SSBGST.

The factorization has the form A = S^T *S where S is a band matrix of the same bandwidth as A and the following structure:

```
S = ( U    )
    ( M  L )
```

where U is upper triangular of order m = (n+kd)/2, and L is lower triangular of order n-m.

Syntax

Fortran specification:

```
use armpl_library

subroutine spbstf(UPLO, N, KD, AB, LDAB, INFO)
```

C specification:

```
#include "armpl.h"

void spbstf_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
             float *ab, const armpl_int_t *ldab, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is REAL

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first kd+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, if INFO = 0, the factor S from the split Cholesky factorization A = S^T *S. See Further Details.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the factorization could not be completed, because the updated element a(i,i) was negative; the matrix A is not positive definite.

Related Information

For this routine in other precisions, please see *cpbstf*, *dpbstf* and *zpbstf*. It also exists with a native C interface as *LAPACKE_spbstf*.

4.12.28 ssbgst

ssbgst reduces a real symmetric-definite banded generalized eigenproblem A*x = lambda*B*x to standard form C*y = lambda*y, such that C has the same bandwidth as A.

B must have been previously factorized as S^T *S by SPBSTF, using a split Cholesky factorization. A is overwritten by C = X^T *A*X, where X = S**(-1)*Q and Q is an orthogonal matrix chosen to preserve the bandwidth of A.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssbgst(VECT, UPLO, N, KA, KB, AB, LDAB, BB, LDBB, X, LDX, WORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void ssbgst_(const char *vect, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *ka, const armpl_int_t *kb, float *ab,
             const armpl_int_t *ldab, const float *bb,
             const armpl_int_t *ldbb, float *x, const armpl_int_t *ldx,
             float *work, armpl_int_t *info, ... );
```

Parameters

VECT Input parameter.

VECT is CHARACTER*1

= 'N': do not form the transformation matrix X; = 'V': form X.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

KA Input parameter.

KA is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KA >= 0.

KB Input parameter.

KB is INTEGER

The number of superdiagonals of the matrix B if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KA >= KB >= 0.

AB Input and output parameter.

AB is REAL

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first ka+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as

follows: if UPLO = 'U', AB(ka+1+i-j,j) = A(i,j) for max(1,j-ka)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+ka).

On exit, the transformed matrix X^T *A*X, stored in the same format as A.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KA+1.

BB Input parameter.

BB is REAL

BB is an array, dimension (LDBB, N). The banded factor S from the split Cholesky factorization of B, as returned by SPBSTF, stored in the first KB+1 rows of the array.

LDBB Input parameter.

LDBB is INTEGER

The leading dimension of the array BB. LDBB >= KB+1.

X Output parameter.

X is REAL

X is an array, dimension (LDX, N). If VECT = 'V', the n-by-n matrix X. If VECT = 'N', the array X is not referenced.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N) if VECT = 'V'; LDX >= 1 otherwise.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *dsbgst*. It also exists with a native C interface as *LAPACKE_ssbgst*.

4.12.29 ssbgv

ssbgv computes all the eigenvalues, and optionally, the eigenvectors of a real generalized symmetric-definite banded eigenproblem, of the form A*x=(lambda)*B*x. Here A and B are assumed to be symmetric and banded, and B is also positive definite.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssbgv(JOBZ, UPLO, N, KA, KB, AB, LDAB, BB, LDBB, W, Z, LDZ, WORK,
                 INFO)
```

C specification:

```
#include "armpl.h"

void ssbgv_(const char *jobz, const char *uplo, const armpl_int_t *n,
            const armpl_int_t *ka, const armpl_int_t *kb, float *ab,
            const armpl_int_t *ldab, float *bb, const armpl_int_t *ldbb,
            float *w, float *z, const armpl_int_t *ldz, float *work,
            armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangles of A and B are stored; = 'L': Lower triangles of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

KA Input parameter.

KA is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KA >= 0.

KB Input parameter.

KB is INTEGER

The number of superdiagonals of the matrix B if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KB >= 0.

AB Input and output parameter.

AB is REAL

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first ka+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(ka+1+i-j,j) = A(i,j) for max(1,j-ka)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+ka).

On exit, the contents of AB are destroyed.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KA+1.

BB Input and output parameter.

BB is REAL

BB is an array, dimension (LDBB, N). On entry, the upper or lower triangle of the symmetric band matrix B, stored in the first kb+1 rows of the array. The j-th column of B is stored in the j-th column of the array BB as follows: if UPLO = 'U', BB(kb+1+i-j,j) = B(i,j) for max(1,j-kb)<=i<=j; if UPLO = 'L', BB(1+i-j,j) = B(i,j) for j<=i<=min(n,j+kb).

On exit, the factor S from the split Cholesky factorization B = S^T *S, as returned by SPBSTF.

LDBB Input parameter.

LDBB is INTEGER

The leading dimension of the array BB. LDBB >= KB+1.

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is REAL

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the matrix Z of eigenvectors, with the i-th column of Z holding the eigenvector associated with W(i). The eigenvectors are normalized so that Z^T *B* Z = I. If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= N.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (3*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is: <= N: the algorithm failed to converge: i off-diagonal elements of an intermediate tridiagonal form did not converge to zero; > N: if INFO = N + i, for 1 <= i <= N, then SPBSTF returned INFO = i: B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *dsbgv*. It also exists with a native C interface as *LAPACKE_ssbgv*.

4.12.30 ssbgvd

ssbgvd computes all the eigenvalues, and optionally, the eigenvectors of a real generalized symmetric-definite banded eigenproblem, of the form A*x=(lambda)*B*x. Here A and B are assumed to be symmetric and banded, and B is also positive definite. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssbgvd(JOBZ, UPLO, N, KA, KB, AB, LDAB, BB, LDBB, W, Z, LDZ, WORK,
                  LWORK, IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ssbgvd_(const char *jobz, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *ka, const armpl_int_t *kb, float *ab,
             const armpl_int_t *ldab, float *bb, const armpl_int_t *ldbb,
             float *w, float *z, const armpl_int_t *ldz, float *work,
             const armpl_int_t *lwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangles of A and B are stored; = 'L': Lower triangles of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

KA Input parameter.

KA is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KA >= 0.

KB Input parameter.

KB is INTEGER

The number of superdiagonals of the matrix B if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KB >= 0.

AB Input and output parameter.

AB is REAL

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first ka+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(ka+1+i-j,j) = A(i,j) for max(1,j-ka)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+ka).

On exit, the contents of AB are destroyed.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KA+1.

BB Input and output parameter.

BB is REAL

BB is an array, dimension (LDBB, N). On entry, the upper or lower triangle of the symmetric band matrix B, stored in the first kb+1 rows of the array. The j-th column of B is stored in the j-th column of the array BB as follows: if UPLO = 'U', BB(ka+1+i-j,j) = B(i,j) for max(1,j-kb)<=i<=j; if UPLO = 'L', BB(1+i-j,j) = B(i,j) for j<=i<=min(n,j+kb).

On exit, the factor S from the split Cholesky factorization B = S^T *S, as returned by SPBSTF.

LDBB Input parameter.

LDBB is INTEGER

The leading dimension of the array BB. LDBB >= KB+1.

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is REAL

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the matrix Z of eigenvectors, with the i-th column of Z holding the eigenvector associated with W(i). The eigenvectors are normalized so Z^T *B* Z = I. If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If N <= 1, LWORK >= 1. If JOBZ = 'N' and N > 1, LWORK >= 3*N. If JOBZ = 'V' and N > 1, LWORK >= 1 + 5* N + 2*N**2.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if LIWORK > 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. If JOBZ = 'N' or N <= 1, LIWORK >= 1. If JOBZ = 'V' and N > 1, LIWORK >= 3 + 5*N.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is: <= N: the algorithm failed to converge: i off-diagonal elements of an intermediate tridiagonal form did not converge to zero; > N: if INFO = N + i, for 1 <= i <= N, then SPBSTF returned INFO = i: B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *dsbgvd*. It also exists with a native C interface as *LAPACKE_ssbgvd*.

4.12.31 ssbgvx

ssbgvx computes selected eigenvalues, and optionally, eigenvectors of a real generalized symmetric-definite banded eigenproblem, of the form A*x=(lambda)*B*x. Here A and B are assumed to be symmetric and banded, and B is also positive definite. Eigenvalues and eigenvectors can be selected by specifying either all eigenvalues, a range of values or a range of indices for the desired eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssbgvx(JOBZ, RANGE, UPLO, N, KA, KB, AB, LDAB, BB, LDBB, Q, LDQ,
                  VL, VU, IL, IU, ABSTOL, M, W, Z, LDZ, WORK, IWORK, IFAIL,
                  INFO)
```

C specification:

```
#include "armpl.h"

void ssbgvx_(const char *jobz, const char *range, const char *uplo,
             const armpl_int_t *n, const armpl_int_t *ka,
             const armpl_int_t *kb, float *ab, const armpl_int_t *ldab,
             float *bb, const armpl_int_t *ldbb, float *q,
             const armpl_int_t *ldq, const float *vl, const float *vu,
             const armpl_int_t *il, const armpl_int_t *iu,
             const float *abstol, armpl_int_t *m, float *w, float *z,
             const armpl_int_t *ldz, float *work, armpl_int_t *iwork,
             armpl_int_t *ifail, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found. = 'V': all eigenvalues in the half-open interval (VL,VU] will be found. = 'I': the IL-th through IU-th eigenvalues will be found.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangles of A and B are stored; = 'L': Lower triangles of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

KA Input parameter.

KA is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KA >= 0.

KB Input parameter.

KB is INTEGER

The number of superdiagonals of the matrix B if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KB >= 0.

AB Input and output parameter.

AB is REAL

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first ka+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(ka+1+i-j,j) = A(i,j) for max(1,j-ka)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+ka).

On exit, the contents of AB are destroyed.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KA+1.

BB Input and output parameter.

BB is REAL

BB is an array, dimension (LDBB, N). On entry, the upper or lower triangle of the symmetric band matrix B, stored in the first kb+1 rows of the array. The j-th column of B is stored in the j-th column of the array BB as follows: if UPLO = 'U', BB(ka+1+i-j,j) = B(i,j) for max(1,j-kb)<=i<=j; if UPLO = 'L', BB(1+i-j,j) = B(i,j) for j<=i<=min(n,j+kb).

On exit, the factor S from the split Cholesky factorization B = S^T *S, as returned by SPBSTF.

LDBB Input parameter.

LDBB is INTEGER

The leading dimension of the array BB. LDBB >= KB+1.

Q Output parameter.

Q is REAL

Q is an array, dimension (LDQ, N). If JOBZ = 'V', the n-by-n matrix used in the reduction of A*x = (lambda)*B*x to standard form, i.e. C*x = (lambda)*x, and consequently C to tridiagonal form. If JOBZ = 'N', the array Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. If JOBZ = 'N', LDQ >= 1. If JOBZ = 'V', LDQ >= max(1, N).

VL Input parameter.

VL is REAL

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is REAL

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is REAL

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing A to tridiagonal form.

Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*SLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*SLAMCH('S').

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is REAL

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the matrix Z of eigenvectors, with the i-th column of Z holding the eigenvector associated with W(i). The eigenvectors are normalized so Z^T *B* Z = I. If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (7*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (5*N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (M)

If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvalues that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0 : successful exit < 0 : if INFO = -i, the i-th argument had an illegal value <= N: if INFO = i, then i eigenvectors failed to converge. Their indices are stored in IFAIL. > N : SPBSTF returned an error code; i.e., if INFO = N + i, for 1 <= i <= N, then the leading minor of order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *dsbgvx*. It also exists with a native C interface as *LAPACKE_ssbgvx*.

4.12.32 sspgst

sspgst reduces a real symmetric-definite generalized eigenproblem to standard form, using packed storage.

If ITYPE = 1, the problem is A*x = lambda*B*x, and A is overwritten by inv(U^T)*A*inv(U) or inv(L)*A*inv(L^T)

If ITYPE = 2 or 3, the problem is A*B*x = lambda*x or B*A*x = lambda*x, and A is overwritten by U*A*U^T or L^T*A*L.

B must have been previously factorized as U^T*U or L*L^T by SPPTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine sspgst(ITYPE, UPLO, N, AP, BP, INFO)
```

C specification:

```
#include "armpl.h"

void sspgst_(const armpl_int_t *itype, const char *uplo, const armpl_int_t *n,
             float *ap, const float *bp, armpl_int_t *info, ... );
```

Parameters

ITYPE Input parameter.

ITYPE is INTEGER

= 1: compute inv(U^T)*A*inv(U) or inv(L)*A*inv(L^T); = 2 or 3: compute U*A*U^T or L^T*A*L.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored and B is factored as U^T*U; = 'L': Lower triangle of A is stored and B is factored as L*L^T .

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

AP Input and output parameter.

AP is REAL

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

On exit, if INFO = 0, the transformed matrix, stored in the same format as A.

BP Input parameter.

BP is REAL

BP is an array, dimension (N*(N+1)/2). The triangular factor from the Cholesky factorization of B, stored in the same format as A, as returned by SPPTRF.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dspgst*. It also exists with a native C interface as *LAPACKE_sspgst*.

4.12.33 sspgv

sspgv computes all the eigenvalues and, optionally, the eigenvectors of a real generalized symmetric-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x. Here A and B are assumed to be symmetric, stored in packed format, and B is also positive definite.

Syntax

Fortran specification:

```
use armpl_library

subroutine sspgv(ITYPE, JOBZ, UPLO, N, AP, BP, W, Z, LDZ, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void sspgv_(const armpl_int_t *itype, const char *jobz, const char *uplo,
            const armpl_int_t *n, float *ap, float *bp, float *w, float *z,
            const armpl_int_t *ldz, float *work, armpl_int_t *info, ... );
```

Parameters

ITYPE Input parameter.

ITYPE is INTEGER

Specifies the problem type to be solved: = 1: A*x = (lambda)*B*x = 2: A*B*x = (lambda)*x = 3: B*A*x = (lambda)*x

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangles of A and B are stored; = 'L': Lower triangles of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

AP Input and output parameter.

AP is REAL

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n.

On exit, the contents of AP are destroyed.

BP Input and output parameter.

BP is REAL

BP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix B, packed columnwise in a linear array. The j-th column of B is stored in the array BP as follows: if UPLO = 'U', BP(i + (j-1)*j/2) = B(i,j) for 1<=i<=j; if UPLO = 'L', BP(i + (j-1)*(2*n-j)/2) = B(i,j) for j<=i<=n.

On exit, the triangular factor U or L from the Cholesky factorization B = U^T *U or B = L*L^T , in the same storage format as B.

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is REAL

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the matrix Z of eigenvectors. The eigenvectors are normalized as follows: if ITYPE = 1 or 2, Z^T *B* Z = I; if ITYPE = 3, Z^T *inv(B)* Z = I. If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (3*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: SPPTRF or SSPEV returned an error code: <= N: if INFO = i, SSPEV failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero. > N: if INFO = n + i, for 1 <= i <= n, then the leading minor of order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *dspgv*. It also exists with a native C interface as *LAPACKE_sspgv*.

4.12.34 sspgvd

`sspgvd` computes all the eigenvalues, and optionally, the eigenvectors of a real generalized symmetric-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x. Here A and B are assumed to be symmetric, stored in packed format, and B is also positive definite. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine sspgvd(ITYPE, JOBZ, UPLO, N, AP, BP, W, Z, LDZ, WORK, LWORK, IWORK,
                  LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sspgvd_(const armpl_int_t *itype, const char *jobz, const char *uplo,
             const armpl_int_t *n, float *ap, float *bp, float *w, float *z,
             const armpl_int_t *ldz, float *work, const armpl_int_t *lwork,
             armpl_int_t *iwork, const armpl_int_t *liwork, armpl_int_t *info,
             ... );
```

Parameters

ITYPE Input parameter.

ITYPE is INTEGER

Specifies the problem type to be solved: = 1: A*x = (lambda)*B*x = 2: A*B*x = (lambda)*x = 3: B*A*x = (lambda)*x

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangles of A and B are stored; = 'L': Lower triangles of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

AP Input and output parameter.

AP is REAL

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n.

On exit, the contents of AP are destroyed.

BP Input and output parameter.

BP is REAL

BP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix B, packed columnwise in a linear array. The j-th column of B is stored in the array BP as follows: if UPLO = 'U', BP(i + (j-1)*j/2) = B(i,j) for 1<=i<=j; if UPLO = 'L', BP(i + (j-1)*(2*n-j)/2) = B(i,j) for j<=i<=n.

On exit, the triangular factor U or L from the Cholesky factorization $B = U^T * U$ or $B = L * L^T$, in the same storage format as B.

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is REAL

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the matrix Z of eigenvectors. The eigenvectors are normalized as follows: if ITYPE = 1 or 2, Z^T *B* Z = I; if ITYPE = 3, Z^T *inv(B)* Z = I. If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the required LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If N <= 1, LWORK >= 1. If JOBZ = 'N' and N > 1, LWORK >= 2*N. If JOBZ = 'V' and N > 1, LWORK >= 1 + 6* N + 2*N**2.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the required sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the required LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. If JOBZ = 'N' or N <= 1, LIWORK >= 1. If JOBZ = 'V' and N > 1, LIWORK >= 3 + 5*N.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the required sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: SPPTRF or SSPEVD returned an error code: <= N: if INFO = i, SSPEVD failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero; > N: if INFO = N + i, for 1 <= i <= N, then the leading minor of order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *dspgvd*. It also exists with a native C interface as *LAPACKE_sspgvd*.

4.12.35 sspgvx

sspgvx computes selected eigenvalues, and optionally, eigenvectors of a real generalized symmetric-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x. Here A and B are assumed to be symmetric, stored in packed storage, and B is also positive definite. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine sspgvx(ITYPE, JOBZ, RANGE, UPLO, N, AP, BP, VL, VU, IL, IU, ABSTOL,
                  M, W, Z, LDZ, WORK, IWORK, IFAIL, INFO)
```

C specification:

```
#include "armpl.h"

void sspgvx_(const armpl_int_t *itype, const char *jobz, const char *range,
             const char *uplo, const armpl_int_t *n, float *ap, float *bp,
             const float *vl, const float *vu, const armpl_int_t *il,
             const armpl_int_t *iu, const float *abstol, armpl_int_t *m,
             float *w, float *z, const armpl_int_t *ldz, float *work,
             armpl_int_t *iwork, armpl_int_t *ifail, armpl_int_t *info,
             ... );
```

Parameters

ITYPE Input parameter.

ITYPE is INTEGER

Specifies the problem type to be solved: = 1: A*x = (lambda)*B*x = 2: A*B*x = (lambda)*x = 3: B*A*x = (lambda)*x

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found. = 'V': all eigenvalues in the half-open interval (VL,VU] will be found. = 'I': the IL-th through IU-th eigenvalues will be found.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A and B are stored; = 'L': Lower triangle of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrix pencil (A,B). N >= 0.

AP Input and output parameter.

AP is REAL

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n.

On exit, the contents of AP are destroyed.

BP Input and output parameter.

BP is REAL

BP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix B, packed columnwise in a linear array. The j-th column of B is stored in the array BP as follows: if UPLO = 'U', BP(i + (j-1)*j/2) = B(i,j) for 1<=i<=j; if UPLO = 'L', BP(i + (j-1)*(2*n-j)/2) = B(i,j) for j<=i<=n.

On exit, the triangular factor U or L from the Cholesky factorization $B = U^T *U$ or $B = L*L^T$, in the same storage format as B.

VL Input parameter.

VL is REAL

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is REAL

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is REAL

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing A to tridiagonal form.

Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*SLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*SLAMCH('S').

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is REAL

W is an array, dimension (N). On normal exit, the first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is REAL

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'N', then Z is not referenced. If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). The eigenvectors are normalized as follows: if ITYPE = 1 or 2, Z^T *B* Z = I; if ITYPE = 3, Z^T *inv(B)* Z = I.

If an eigenvector fails to converge, then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (8*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (5*N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (N)

If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: SPPTRF or SSPEVX returned an error code: <= N: if INFO = i, SSPEVX failed to converge; i eigenvectors failed to converge. Their indices are stored in array IFAIL. > N: if INFO = N + i, for 1 <= i <= N, then the leading minor of order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *dspgvx*. It also exists with a native C interface as *LAPACKE_sspgvx*.

4.12.36 ssygst

`ssygst` reduces a real symmetric-definite generalized eigenproblem to standard form.

If ITYPE = 1, the problem is A*x = lambda*B*x, and A is overwritten by inv(U^T)*A*inv(U) or inv(L)*A*inv(L^T)

If ITYPE = 2 or 3, the problem is A*B*x = lambda*x or B*A*x = lambda*x, and A is overwritten by U*A*U^T or L^T*A*L.

B must have been previously factorized as U^T*U or L*L^T by SPOTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssygst(ITYPE, UPLO, N, A, LDA, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void ssygst_(const armpl_int_t *itype, const char *uplo, const armpl_int_t *n,
             float *a, const armpl_int_t *lda, const float *b,
             const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

ITYPE Input parameter.

`ITYPE` is `INTEGER`

= 1: compute inv(U^T)*A*inv(U) or inv(L)*A*inv(L^T); = 2 or 3: compute U*A*U^T or L^T*A*L.

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

= '`U`': Upper triangle of `A` is stored and `B` is factored as U^T*U; = '`L`': Lower triangle of `A` is stored and `B` is factored as L*L^T.

N Input parameter.

`N` is `INTEGER`

The order of the matrices `A` and `B`. `N` >= 0.

A Input and output parameter.

`A` is `REAL`

`A` is an array, dimension (`LDA`, `N`). On entry, the symmetric matrix `A`. If `UPLO` = '`U`', the leading N-by-N upper triangular part of `A` contains the upper triangular part of the matrix `A`, and the strictly lower triangular part of `A` is not referenced. If `UPLO` = '`L`', the leading N-by-N lower triangular part of `A` contains the lower triangular part of the matrix `A`, and the strictly upper triangular part of `A` is not referenced.

On exit, if `INFO` = 0, the transformed matrix, stored in the same format as `A`.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input parameter.

B is REAL

B is an array, dimension (LDB, N). The triangular factor from the Cholesky factorization of B, as returned by SPOTRF.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dsygst*. It also exists with a native C interface as *LAPACKE_ssygst*.

4.12.37 ssygv

ssygv computes all the eigenvalues, and optionally, the eigenvectors of a real generalized symmetric-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x. Here A and B are assumed to be symmetric and B is also positive definite.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssygv(ITYPE, JOBZ, UPLO, N, A, LDA, B, LDB, W, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ssygv_(const armpl_int_t *itype, const char *jobz, const char *uplo,
            const armpl_int_t *n, float *a, const armpl_int_t *lda, float *b,
            const armpl_int_t *ldb, float *w, float *work,
            const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

ITYPE Input parameter.

ITYPE is INTEGER

Specifies the problem type to be solved: = 1: A*x = (lambda)*B*x = 2: A*B*x = (lambda)*x = 3: B*A*x = (lambda)*x

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangles of A and B are stored; = 'L': Lower triangles of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A.

On exit, if JOBZ = 'V', then if INFO = 0, A contains the matrix Z of eigenvectors. The eigenvectors are normalized as follows: if ITYPE = 1 or 2, Z^T *B*Z = I; if ITYPE = 3, Z^T *inv(B)*Z = I. If JOBZ = 'N', then on exit the upper triangle (if UPLO='U') or the lower triangle (if UPLO='L') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, N). On entry, the symmetric positive definite matrix B. If UPLO = 'U', the leading N-by-N upper triangular part of B contains the upper triangular part of the matrix B. If UPLO = 'L', the leading N-by-N lower triangular part of B contains the lower triangular part of the matrix B.

On exit, if INFO <= N, the part of B containing the matrix is overwritten by the triangular factor U or L from the Cholesky factorization B = U^T *U or B = L*L^T .

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= max(1,3*N-1). For optimal efficiency, LWORK >= (NB+2)*N, where NB is the blocksize for SSYTRD returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: SPOTRF or SSYEV returned an error code: <= N: if INFO = i, SSYEV failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero; > N: if INFO = N + i, for 1 <= i <= N, then the leading minor of order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *dsygv*. It also exists with a native C interface as *LAPACKE_ssygv*.

4.12.38 ssygvd

ssygvd computes all the eigenvalues, and optionally, the eigenvectors of a real generalized symmetric-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x. Here A and B are assumed to be symmetric and B is also positive definite. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssygvd(ITYPE, JOBZ, UPLO, N, A, LDA, B, LDB, W, WORK, LWORK, IWORK,
                  LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ssygvd_(const armpl_int_t *itype, const char *jobz, const char *uplo,
             const armpl_int_t *n, float *a, const armpl_int_t *lda, float *b,
             const armpl_int_t *ldb, float *w, float *work,
             const armpl_int_t *lwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

ITYPE Input parameter.

ITYPE is INTEGER

Specifies the problem type to be solved: = 1: A*x = (lambda)*B*x = 2: A*B*x = (lambda)*x = 3: B*A*x = (lambda)*x

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangles of A and B are stored; = 'L': Lower triangles of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A.

On exit, if JOBZ = 'V', then if INFO = 0, A contains the matrix Z of eigenvectors. The eigenvectors are normalized as follows: if ITYPE = 1 or 2, Z^T *B*Z = I; if ITYPE = 3, Z^T *inv(B)*Z = I. If JOBZ = 'N', then on exit the upper triangle (if UPLO='U') or the lower triangle (if UPLO='L') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, N). On entry, the symmetric matrix B. If UPLO = 'U', the leading N-by-N upper triangular part of B contains the upper triangular part of the matrix B. If UPLO = 'L', the leading N-by-N lower triangular part of B contains the lower triangular part of the matrix B.

On exit, if INFO <= N, the part of B containing the matrix is overwritten by the triangular factor U or L from the Cholesky factorization B = U^T *U or B = L*L^T .

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If N <= 1, LWORK >= 1. If JOBZ = 'N' and N > 1, LWORK >= 2*N+1. If JOBZ = 'V' and N > 1, LWORK >= 1 + 6* N + 2*N**2.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. If N <= 1, LIWORK >= 1. If JOBZ = 'N' and N > 1, LIWORK >= 1. If JOBZ = 'V' and N > 1, LIWORK >= 3 + 5*N.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: SPOTRF or SSYEVD returned an error code: <= N: if INFO = i and JOBZ = 'N', then the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero; if INFO = i and JOBZ = 'V', then the algorithm failed to compute an eigenvalue while working on the submatrix lying in rows and columns INFO/(N+1) through mod(INFO,N+1); > N: if INFO = N + i, for 1 <= i <= N, then the leading minor of order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *dsygvd*. It also exists with a native C interface as *LAPACKE_ssygvd*.

4.12.39 ssygvx

ssygvx computes selected eigenvalues, and optionally, eigenvectors of a real generalized symmetric-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x. Here A and B are assumed to be symmetric and B is also positive definite. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssygvx(ITYPE, JOBZ, RANGE, UPLO, N, A, LDA, B, LDB, VL, VU, IL, IU,
                  ABSTOL, M, W, Z, LDZ, WORK, LWORK, IWORK, IFAIL, INFO)
```

C specification:

```
#include "armpl.h"

void ssygvx_(const armpl_int_t *itype, const char *jobz, const char *range,
             const char *uplo, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, float *b, const armpl_int_t *ldb,
             const float *vl, const float *vu, const armpl_int_t *il,
```

(continues on next page)

(continued from previous page)

```
                const armpl_int_t *iu, const float *abstol, armpl_int_t *m,
                float *w, float *z, const armpl_int_t *ldz, float *work,
                const armpl_int_t *lwork, armpl_int_t *iwork, armpl_int_t *ifail,
                armpl_int_t *info, ... );
```

Parameters

ITYPE Input parameter.

ITYPE is INTEGER

Specifies the problem type to be solved: = 1: A*x = (lambda)*B*x = 2: A*B*x = (lambda)*x = 3: B*A*x = (lambda)*x

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found. = 'V': all eigenvalues in the half-open interval (VL,VU] will be found. = 'I': the IL-th through IU-th eigenvalues will be found.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A and B are stored; = 'L': Lower triangle of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrix pencil (A, B). N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A.

On exit, the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, N). On entry, the symmetric matrix B. If UPLO = 'U', the leading N-by-N upper triangular part of B contains the upper triangular part of the matrix B. If UPLO = 'L', the leading N-by-N lower triangular part of B contains the lower triangular part of the matrix B.

On exit, if INFO <= N, the part of B containing the matrix is overwritten by the triangular factor U or L from the Cholesky factorization B = U^T *U or B = L*L^T .

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

VL Input parameter.

VL is REAL

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is REAL

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is REAL

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing C to tridiagonal form, where C is the symmetric matrix of the standard symmetric problem to which the generalized problem is transformed.

Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*DLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*SLAMCH('S').

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is REAL

W is an array, dimension (N). On normal exit, the first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is REAL

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'N', then Z is not referenced. If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). The eigenvectors are normalized as follows: if ITYPE = 1 or 2, Z^T *B* Z = I; if ITYPE = 3, Z^T *inv(B)* Z = I.

If an eigenvector fails to converge, then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= max(1,8*N). For optimal efficiency, LWORK >= (NB+3)*N, where NB is the blocksize for SSYTRD returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (5*N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (N)

If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: SPOTRF or SSYEVX returned an error code: <= N: if INFO = i, SSYEVX failed to converge; i eigenvectors failed to converge. Their indices are stored in array IFAIL. > N: if INFO = N + i, for 1 <= i <= N, then the leading minor of order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *dsygvx*. It also exists with a native C interface as *LAPACKE_ssygvx*.

4.12.40 zhbgst

zhbgst reduces a complex Hermitian-definite banded generalized eigenproblem A*x = lambda*B*x to standard form C*y = lambda*y, such that C has the same bandwidth as A.

B must have been previously factorized as S^H *S by ZPBSTF, using a split Cholesky factorization. A is overwritten by C = X^H *A*X, where X = S**(-1)*Q and Q is a unitary matrix chosen to preserve the bandwidth of A.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhbgst(VECT, UPLO, N, KA, KB, AB, LDAB, BB, LDBB, X, LDX, WORK,
                  RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhbgst_(const char *vect, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *ka, const armpl_int_t *kb,
             armpl_doublecomplex_t *ab, const armpl_int_t *ldab,
             const armpl_doublecomplex_t *bb, const armpl_int_t *ldbb,
             armpl_doublecomplex_t *x, const armpl_int_t *ldx,
             armpl_doublecomplex_t *work, double *rwork, armpl_int_t *info,
             ... );
```

Parameters

VECT Input parameter.

VECT is CHARACTER*1

= 'N': do not form the transformation matrix X; = 'V': form X.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

KA Input parameter.

KA is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KA >= 0.

KB Input parameter.

KB is INTEGER

The number of superdiagonals of the matrix B if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KA >= KB >= 0.

AB Input and output parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the Hermitian band matrix A, stored in the first ka+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(ka+1+i-j,j) = A(i,j) for max(1,j-ka)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+ka).

On exit, the transformed matrix X^H *A*X, stored in the same format as A.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KA+1.

BB Input parameter.

BB is COMPLEX*16

BB is an array, dimension (LDBB, N). The banded factor S from the split Cholesky factorization of B, as returned by ZPBSTF, stored in the first kb+1 rows of the array.

LDBB Input parameter.

LDBB is INTEGER

The leading dimension of the array BB. LDBB >= KB+1.

X Output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, N). If VECT = 'V', the n-by-n matrix X. If VECT = 'N', the array X is not referenced.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, N) if VECT = 'V'; LDX >= 1 otherwise.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *chbgst*. It also exists with a native C interface as *LAPACKE_zhbgst*.

4.12.41 zhbgv

zhbgv computes all the eigenvalues, and optionally, the eigenvectors of a complex generalized Hermitian-definite banded eigenproblem, of the form A*x=(lambda)*B*x. Here A and B are assumed to be Hermitian and banded, and B is also positive definite.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhbgv(JOBZ, UPLO, N, KA, KB, AB, LDAB, BB, LDBB, W, Z, LDZ, WORK,
                 RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhbgv_(const char *jobz, const char *uplo, const armpl_int_t *n,
            const armpl_int_t *ka, const armpl_int_t *kb,
            armpl_doublecomplex_t *ab, const armpl_int_t *ldab,
            armpl_doublecomplex_t *bb, const armpl_int_t *ldbb, double *w,
            armpl_doublecomplex_t *z, const armpl_int_t *ldz,
            armpl_doublecomplex_t *work, double *rwork, armpl_int_t *info,
            ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangles of A and B are stored; = 'L': Lower triangles of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

KA Input parameter.

KA is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KA >= 0.

KB Input parameter.

KB is INTEGER

The number of superdiagonals of the matrix B if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KB >= 0.

AB Input and output parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the Hermitian band matrix A, stored in the first ka+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(ka+1+i-j,j) = A(i,j) for max(1,j-ka)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+ka).

On exit, the contents of AB are destroyed.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KA+1.

BB Input and output parameter.

BB is COMPLEX*16

BB is an array, dimension (LDBB, N). On entry, the upper or lower triangle of the Hermitian band matrix B, stored in the first kb+1 rows of the array. The j-th column of B is stored in the j-th column of the array BB as

follows: if UPLO = 'U', BB(kb+1+i-j,j) = B(i,j) for max(1,j-kb)<=i<=j; if UPLO = 'L', BB(1+i-j,j) = B(i,j) for j<=i<=min(n,j+kb).

On exit, the factor S from the split Cholesky factorization B = S^H *S, as returned by ZPBSTF.

LDBB Input parameter.

LDBB is INTEGER

The leading dimension of the array BB. LDBB >= KB+1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the matrix Z of eigenvectors, with the i-th column of Z holding the eigenvector associated with W(i). The eigenvectors are normalized so that Z^H *B* Z = I. If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= N.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (3*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is: <= N: the algorithm failed to converge: i off-diagonal elements of an intermediate tridiagonal form did not converge to zero; > N: if INFO = N + i, for 1 <= i <= N, then ZPBSTF returned INFO = i: B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *chbgv*. It also exists with a native C interface as *LAPACKE_zhbgv*.

4.12.42 zhbgvd

zhbgvd computes all the eigenvalues, and optionally, the eigenvectors of a complex generalized Hermitian-definite banded eigenproblem, of the form A*x=(lambda)*B*x. Here A and B are assumed to be Hermitian and banded, and B is also positive definite. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhbgvd(JOBZ, UPLO, N, KA, KB, AB, LDAB, BB, LDBB, W, Z, LDZ, WORK,
                  LWORK, RWORK, LRWORK, IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhbgvd_(const char *jobz, const char *uplo, const armpl_int_t *n,
             const armpl_int_t *ka, const armpl_int_t *kb,
             armpl_doublecomplex_t *ab, const armpl_int_t *ldab,
             armpl_doublecomplex_t *bb, const armpl_int_t *ldbb, double *w,
             armpl_doublecomplex_t *z, const armpl_int_t *ldz,
             armpl_doublecomplex_t *work, const armpl_int_t *lwork,
             double *rwork, const armpl_int_t *lrwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangles of A and B are stored; = 'L': Lower triangles of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

KA Input parameter.

KA is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KA >= 0.

KB Input parameter.

KB is INTEGER

The number of superdiagonals of the matrix B if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KB >= 0.

AB Input and output parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the Hermitian band matrix A, stored in the first ka+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(ka+1+i-j,j) = A(i,j) for max(1,j-ka)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+ka).

On exit, the contents of AB are destroyed.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KA+1.

BB Input and output parameter.

BB is COMPLEX*16

BB is an array, dimension (LDBB, N). On entry, the upper or lower triangle of the Hermitian band matrix B, stored in the first kb+1 rows of the array. The j-th column of B is stored in the j-th column of the array BB as follows: if UPLO = 'U', BB(kb+1+i-j,j) = B(i,j) for max(1,j-kb)<=i<=j; if UPLO = 'L', BB(1+i-j,j) = B(i,j) for j<=i<=min(n,j+kb).

On exit, the factor S from the split Cholesky factorization B = S^H *S, as returned by ZPBSTF.

LDBB Input parameter.

LDBB is INTEGER

The leading dimension of the array BB. LDBB >= KB+1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the matrix Z of eigenvectors, with the i-th column of Z holding the eigenvector associated with W(i). The eigenvectors are normalized so that Z^H *B* Z = I. If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= N.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO=0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If N <= 1, LWORK >= 1. If JOBZ = 'N' and N > 1, LWORK >= N. If JOBZ = 'V' and N > 1, LWORK >= 2*N**2.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (MAX(1, LRWORK)). On exit, if INFO=0, RWORK(1) returns the optimal LRWORK.

LRWORK Input parameter.

LRWORK is INTEGER

The dimension of array RWORK. If N <= 1, LRWORK >= 1. If JOBZ = 'N' and N > 1, LRWORK >= N. If JOBZ = 'V' and N > 1, LRWORK >= 1 + 5* N + 2*N**2.

If LRWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO=0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of array IWORK. If JOBZ = 'N' or N <= 1, LIWORK >= 1. If JOBZ = 'V' and N > 1, LIWORK >= 3 + 5*N.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is: <= N: the algorithm failed to converge: i off-diagonal elements of an intermediate tridiagonal form did not converge to zero; > N: if INFO = N + i, for 1 <= i <= N, then ZPBSTF returned INFO = i: B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *chbgvd*. It also exists with a native C interface as *LAPACKE_zhbgvd*.

4.12.43 zhbgvx

zhbgvx computes all the eigenvalues, and optionally, the eigenvectors of a complex generalized Hermitian-definite banded eigenproblem, of the form A*x=(lambda)*B*x. Here A and B are assumed to be Hermitian and banded, and B is also positive definite. Eigenvalues and eigenvectors can be selected by specifying either all eigenvalues, a range of values or a range of indices for the desired eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhbgvx(JOBZ, RANGE, UPLO, N, KA, KB, AB, LDAB, BB, LDBB, Q, LDQ,
                  VL, VU, IL, IU, ABSTOL, M, W, Z, LDZ, WORK, RWORK, IWORK,
                  IFAIL, INFO)
```

C specification:

```
#include "armpl.h"

void zhbgvx_(const char *jobz, const char *range, const char *uplo,
             const armpl_int_t *n, const armpl_int_t *ka,
             const armpl_int_t *kb, armpl_doublecomplex_t *ab,
             const armpl_int_t *ldab, armpl_doublecomplex_t *bb,
             const armpl_int_t *ldbb, armpl_doublecomplex_t *q,
             const armpl_int_t *ldq, const double *vl, const double *vu,
             const armpl_int_t *il, const armpl_int_t *iu,
```

(continues on next page)

(continued from previous page)

```
                const double *abstol, armpl_int_t *m, double *w,
                armpl_doublecomplex_t *z, const armpl_int_t *ldz,
                armpl_doublecomplex_t *work, double *rwork, armpl_int_t *iwork,
                armpl_int_t *ifail, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found; = 'V': all eigenvalues in the half-open interval (VL,VU] will be found; = 'I': the IL-th through IU-th eigenvalues will be found.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangles of A and B are stored; = 'L': Lower triangles of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

KA Input parameter.

KA is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KA >= 0.

KB Input parameter.

KB is INTEGER

The number of superdiagonals of the matrix B if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KB >= 0.

AB Input and output parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the Hermitian band matrix A, stored in the first ka+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(ka+1+i-j,j) = A(i,j) for max(1,j-ka)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+ka).

On exit, the contents of AB are destroyed.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KA+1.

BB Input and output parameter.

BB is COMPLEX*16

BB is an array, dimension (LDBB, N). On entry, the upper or lower triangle of the Hermitian band matrix B, stored in the first kb+1 rows of the array. The j-th column of B is stored in the j-th column of the array BB as

follows: if UPLO = 'U', BB(kb+1+i-j,j) = B(i,j) for max(1,j-kb)<=i<=j; if UPLO = 'L', BB(1+i-j,j) = B(i,j) for j<=i<=min(n,j+kb).

On exit, the factor S from the split Cholesky factorization B = S^H *S, as returned by ZPBSTF.

LDBB Input parameter.

LDBB is INTEGER

The leading dimension of the array BB. LDBB >= KB+1.

Q Output parameter.

Q is COMPLEX*16

Q is an array, dimension (LDQ, N). If JOBZ = 'V', the n-by-n matrix used in the reduction of A*x = (lambda)*B*x to standard form, i.e. C*x = (lambda)*x, and consequently C to tridiagonal form. If JOBZ = 'N', the array Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. If JOBZ = 'N', LDQ >= 1. If JOBZ = 'V', LDQ >= max(1, N).

VL Input parameter.

VL is DOUBLE PRECISION

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is DOUBLE PRECISION

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is DOUBLE PRECISION

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing AP to tridiagonal form.

Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*DLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*DLAMCH('S').

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the matrix Z of eigenvectors, with the i-th column of Z holding the eigenvector associated with W(i). The eigenvectors are normalized so that Z^H *B* Z = I. If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= N.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (7*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (5*N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (N)

If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, and i is: <= N: then i eigenvectors failed to converge. Their indices are stored in array IFAIL. > N: if INFO = N + i, for 1 <= i <= N, then ZPBSTF returned INFO = i: B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *chbgvx*. It also exists with a native C interface as *LAPACKE_zhbgvx*.

4.12.44 zhegst

zhegst reduces a complex Hermitian-definite generalized eigenproblem to standard form.

If ITYPE = 1, the problem is A*x = lambda*B*x, and A is overwritten by inv(U^H)*A*inv(U) or inv(L)*A*inv(L^H)

If ITYPE = 2 or 3, the problem is A*B*x = lambda*x or B*A*x = lambda*x, and A is overwritten by U*A*U^H or L^H *A*L.

B must have been previously factorized as U^H *U or L*L^H by ZPOTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhegst(ITYPE, UPLO, N, A, LDA, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void zhegst_(const armpl_int_t *itype, const char *uplo, const armpl_int_t *n,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_doublecomplex_t *b, const armpl_int_t *ldb,
             armpl_int_t *info, ... );
```

Parameters

ITYPE Input parameter.

ITYPE is INTEGER

= 1: compute inv(U^H)*A*inv(U) or inv(L)*A*inv(L^H); = 2 or 3: compute U*A*U^H or L^H *A*L.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored and B is factored as U^H *U; = 'L': Lower triangle of A is stored and B is factored as L*L^H .

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the transformed matrix, stored in the same format as A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). The triangular factor from the Cholesky factorization of B, as returned by ZPOTRF.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *chegst*. It also exists with a native C interface as *LAPACKE_zhegst*.

4.12.45 zhegv

zhegv computes all the eigenvalues, and optionally, the eigenvectors of a complex generalized Hermitian-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x. Here A and B are assumed to be Hermitian and B is also positive definite.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhegv(ITYPE, JOBZ, UPLO, N, A, LDA, B, LDB, W, WORK, LWORK, RWORK,
                 INFO)
```

C specification:

```
#include "armpl.h"

void zhegv_(const armpl_int_t *itype, const char *jobz, const char *uplo,
            const armpl_int_t *n, armpl_doublecomplex_t *a,
            const armpl_int_t *lda, armpl_doublecomplex_t *b,
            const armpl_int_t *ldb, double *w, armpl_doublecomplex_t *work,
            const armpl_int_t *lwork, double *rwork, armpl_int_t *info,
            ... );
```

Parameters

ITYPE Input parameter.

ITYPE is INTEGER

Specifies the problem type to be solved: = 1: A*x = (lambda)*B*x = 2: A*B*x = (lambda)*x = 3: B*A*x = (lambda)*x

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangles of A and B are stored; = 'L': Lower triangles of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A.

On exit, if JOBZ = 'V', then if INFO = 0, A contains the matrix Z of eigenvectors. The eigenvectors are normalized as follows: if ITYPE = 1 or 2, Z^H *B*Z = I; if ITYPE = 3, Z^H *inv(B)*Z = I. If JOBZ = 'N', then on exit the upper triangle (if UPLO='U') or the lower triangle (if UPLO='L') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). On entry, the Hermitian positive definite matrix B. If UPLO = 'U', the leading N-by-N upper triangular part of B contains the upper triangular part of the matrix B. If UPLO = 'L', the leading N-by-N lower triangular part of B contains the lower triangular part of the matrix B.

On exit, if INFO <= N, the part of B containing the matrix is overwritten by the triangular factor U or L from the Cholesky factorization B = U^H *U or B = L*L^H .

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= max(1,2*N-1). For optimal efficiency, LWORK >= (NB+1)*N, where NB is the blocksize for ZHETRD returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (max(1, 3*N-2)) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: ZPOTRF or ZHEEV returned an error code: <= N: if INFO = i, ZHEEV failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero; > N: if INFO = N + i, for 1 <= i <= N, then the leading minor of

order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *chegv*. It also exists with a native C interface as *LAPACKE_zhegv*.

4.12.46 zhegvd

zhegvd computes all the eigenvalues, and optionally, the eigenvectors of a complex generalized Hermitian-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x. Here A and B are assumed to be Hermitian and B is also positive definite. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhegvd(ITYPE, JOBZ, UPLO, N, A, LDA, B, LDB, W, WORK, LWORK, RWORK,
                  LRWORK, IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhegvd_(const armpl_int_t *itype, const char *jobz, const char *uplo,
             const armpl_int_t *n, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, double *w, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, double *rwork,
             const armpl_int_t *lrwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

ITYPE Input parameter.

ITYPE is INTEGER

Specifies the problem type to be solved: = 1: A*x = (lambda)*B*x = 2: A*B*x = (lambda)*x = 3: B*A*x = (lambda)*x

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangles of A and B are stored; = 'L': Lower triangles of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A.

On exit, if JOBZ = 'V', then if INFO = 0, A contains the matrix Z of eigenvectors. The eigenvectors are normalized as follows: if ITYPE = 1 or 2, Z^H *B*Z = I; if ITYPE = 3, Z^H *inv(B)*Z = I. If JOBZ = 'N', then on exit the upper triangle (if UPLO='U') or the lower triangle (if UPLO='L') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). On entry, the Hermitian matrix B. If UPLO = 'U', the leading N-by-N upper triangular part of B contains the upper triangular part of the matrix B. If UPLO = 'L', the leading N-by-N lower triangular part of B contains the lower triangular part of the matrix B.

On exit, if INFO <= N, the part of B containing the matrix is overwritten by the triangular factor U or L from the Cholesky factorization B = U^H *U or B = L*L^H .

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. If N <= 1, LWORK >= 1. If JOBZ = 'N' and N > 1, LWORK >= N + 1. If JOBZ = 'V' and N > 1, LWORK >= 2* N + N**2.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (MAX(1, LRWORK)). On exit, if INFO = 0, RWORK(1) returns the optimal LRWORK.

LRWORK Input parameter.

LRWORK is INTEGER

The dimension of the array RWORK. If N <= 1, LRWORK >= 1. If JOBZ = 'N' and N > 1, LRWORK >= N. If JOBZ = 'V' and N > 1, LRWORK >= 1 + 5* N + 2*N**2.

If LRWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. If N <= 1, LIWORK >= 1. If JOBZ = 'N' and N > 1, LIWORK >= 1. If JOBZ = 'V' and N > 1, LIWORK >= 3 + 5*N.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: ZPOTRF or ZHEEVD returned an error code: <= N: if INFO = i and JOBZ = 'N', then the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero; if INFO = i and JOBZ = 'V', then the algorithm failed to compute an eigenvalue while working on the submatrix lying in rows and columns INFO/(N+1) through mod(INFO,N+1); > N: if INFO = N + i, for 1 <= i <= N, then the leading minor of order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *chegvd*. It also exists with a native C interface as *LAPACKE_zhegvd*.

4.12.47 zhegvx

zhegvx computes selected eigenvalues, and optionally, eigenvectors of a complex generalized Hermitian-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x. Here A and B are assumed to be Hermitian and B is also positive definite. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhegvx(ITYPE, JOBZ, RANGE, UPLO, N, A, LDA, B, LDB, VL, VU, IL, IU,
                  ABSTOL, M, W, Z, LDZ, WORK, LWORK, RWORK, IWORK, IFAIL,
                  INFO)
```

C specification:

```
#include "armpl.h"

void zhegvx_(const armpl_int_t *itype, const char *jobz, const char *range,
             const char *uplo, const armpl_int_t *n, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, const double *vl, const double *vu,
             const armpl_int_t *il, const armpl_int_t *iu,
             const double *abstol, armpl_int_t *m, double *w,
             armpl_doublecomplex_t *z, const armpl_int_t *ldz,
             armpl_doublecomplex_t *work, const armpl_int_t *lwork,
             double *rwork, armpl_int_t *iwork, armpl_int_t *ifail,
             armpl_int_t *info, ... );
```

Parameters

ITYPE Input parameter.

ITYPE is INTEGER

Specifies the problem type to be solved: = 1: A*x = (lambda)*B*x = 2: A*B*x = (lambda)*x = 3: B*A*x = (lambda)*x

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found. = 'V': all eigenvalues in the half-open interval (VL,VU] will be found. = 'I': the IL-th through IU-th eigenvalues will be found.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangles of A and B are stored; = 'L': Lower triangles of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A.

On exit, the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). On entry, the Hermitian matrix B. If UPLO = 'U', the leading N-by-N upper triangular part of B contains the upper triangular part of the matrix B. If UPLO = 'L', the leading N-by-N lower triangular part of B contains the lower triangular part of the matrix B.

On exit, if INFO <= N, the part of B containing the matrix is overwritten by the triangular factor U or L from the Cholesky factorization B = U^H *U or B = L*L^H .

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

VL Input parameter.

VL is DOUBLE PRECISION

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is DOUBLE PRECISION

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is DOUBLE PRECISION

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing C to tridiagonal form, where C is the symmetric matrix of the standard symmetric problem to which the generalized problem is transformed.

Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*DLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*DLAMCH('S').

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). The first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'N', then Z is not referenced. If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). The eigenvectors are normalized as follows: if ITYPE = 1 or 2, Z^T *B* Z = I; if ITYPE = 3, Z^T *inv(B)* Z = I.

If an eigenvector fails to converge, then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= max(1,2*N). For optimal efficiency, LWORK >= (NB+1)*N, where NB is the blocksize for ZHETRD returned by ILAENV.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (7*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (5*N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (N)

If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: ZPOTRF or ZHEEVX returned an error code: <= N: if INFO = i, ZHEEVX failed to converge; i eigenvectors failed to converge. Their indices are stored in array IFAIL. > N: if INFO = N + i, for 1 <= i <= N, then the leading minor of order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *chegvx*. It also exists with a native C interface as *LAPACKE_zhegvx*.

4.12.48 zhpgst

`zhpgst` reduces a complex Hermitian-definite generalized eigenproblem to standard form, using packed storage.

If ITYPE = 1, the problem is A*x = lambda*B*x, and A is overwritten by inv(U^H)*A*inv(U) or inv(L)*A*inv(L^H)

If ITYPE = 2 or 3, the problem is A*B*x = lambda*x or B*A*x = lambda*x, and A is overwritten by U*A*U^H or L^H *A*L.

B must have been previously factorized as U^H *U or L*L^H by ZPPTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhpgst(ITYPE, UPLO, N, AP, BP, INFO)
```

C specification:

```
#include "armpl.h"

void zhpgst_(const armpl_int_t *itype, const char *uplo, const armpl_int_t *n,
             armpl_doublecomplex_t *ap, const armpl_doublecomplex_t *bp,
             armpl_int_t *info, ... );
```

Parameters

ITYPE Input parameter.

`ITYPE` is `INTEGER`

= 1: compute inv(U^H)*A*inv(U) or inv(L)*A*inv(L^H); = 2 or 3: compute U*A*U^H or L^H *A*L.

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

= '`U`': Upper triangle of A is stored and B is factored as U^H *U; = '`L`': Lower triangle of A is stored and B is factored as L*L^H .

N Input parameter.

`N` is `INTEGER`

The order of the matrices A and B. `N` >= 0.

AP Input and output parameter.

`AP` is `COMPLEX*16`

`AP` is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the Hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array `AP` as follows: if `UPLO` = '`U`', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if `UPLO` = '`L`', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

On exit, if `INFO` = 0, the transformed matrix, stored in the same format as A.

BP Input parameter.

`BP` is `COMPLEX*16`

`BP` is an array, dimension (N*(N+1)/2). The triangular factor from the Cholesky factorization of B, stored in the same format as A, as returned by ZPPTRF.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *chpgst*. It also exists with a native C interface as *LAPACKE_zhpgst*.

4.12.49 zhpgv

zhpgv computes all the eigenvalues and, optionally, the eigenvectors of a complex generalized Hermitian-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x. Here A and B are assumed to be Hermitian, stored in packed format, and B is also positive definite.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhpgv(ITYPE, JOBZ, UPLO, N, AP, BP, W, Z, LDZ, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhpgv_(const armpl_int_t *itype, const char *jobz, const char *uplo,
            const armpl_int_t *n, armpl_doublecomplex_t *ap,
            armpl_doublecomplex_t *bp, double *w, armpl_doublecomplex_t *z,
            const armpl_int_t *ldz, armpl_doublecomplex_t *work,
            double *rwork, armpl_int_t *info, ... );
```

Parameters

ITYPE Input parameter.

ITYPE is INTEGER

Specifies the problem type to be solved: = 1: A*x = (lambda)*B*x = 2: A*B*x = (lambda)*x = 3: B*A*x = (lambda)*x

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangles of A and B are stored; = 'L': Lower triangles of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

AP Input and output parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the Hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n.

On exit, the contents of AP are destroyed.

BP Input and output parameter.

BP is COMPLEX*16

BP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the Hermitian matrix B, packed columnwise in a linear array. The j-th column of B is stored in the array BP as follows: if UPLO = 'U', BP(i + (j-1)*j/2) = B(i,j) for 1<=i<=j; if UPLO = 'L', BP(i + (j-1)*(2*n-j)/2) = B(i,j) for j<=i<=n.

On exit, the triangular factor U or L from the Cholesky factorization B = U^H *U or B = L*L^H , in the same storage format as B.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the matrix Z of eigenvectors. The eigenvectors are normalized as follows: if ITYPE = 1 or 2, Z^H *B* Z = I; if ITYPE = 3, Z^H *inv(B)* Z = I. If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (max(1, 2*N-1)) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (max(1, 3*N-2)) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: ZPPTRF or ZHPEV returned an error code: <= N: if INFO = i, ZHPEV failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not convergeto zero; > N: if INFO = N + i, for 1 <= i <= n, then the leading minor of order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *chpgv*. It also exists with a native C interface as *LAPACKE_zhpgv*.

4.12.50 zhpgvd

`zhpgvd` computes all the eigenvalues and, optionally, the eigenvectors of a complex generalized Hermitian-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x. Here A and B are assumed to be Hermitian, stored in packed format, and B is also positive definite. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhpgvd(ITYPE, JOBZ, UPLO, N, AP, BP, W, Z, LDZ, WORK, LWORK, RWORK,
                  LRWORK, IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhpgvd_(const armpl_int_t *itype, const char *jobz, const char *uplo,
             const armpl_int_t *n, armpl_doublecomplex_t *ap,
             armpl_doublecomplex_t *bp, double *w, armpl_doublecomplex_t *z,
             const armpl_int_t *ldz, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, double *rwork,
             const armpl_int_t *lrwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

ITYPE Input parameter.

`ITYPE` is `INTEGER`

Specifies the problem type to be solved: = 1: A*x = (lambda)*B*x = 2: A*B*x = (lambda)*x = 3: B*A*x = (lambda)*x

JOBZ Input parameter.

`JOBZ` is `CHARACTER*1`

= '`N`': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

= '`U`': Upper triangles of A and B are stored; = '`L`': Lower triangles of A and B are stored.

N Input parameter.

`N` is `INTEGER`

The order of the matrices A and B. `N` >= 0.

AP Input and output parameter.

`AP` is `COMPLEX*16`

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the Hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n.

On exit, the contents of AP are destroyed.

BP Input and output parameter.

BP is COMPLEX*16

BP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the Hermitian matrix B, packed columnwise in a linear array. The j-th column of B is stored in the array BP as follows: if UPLO = 'U', BP(i + (j-1)*j/2) = B(i,j) for 1<=i<=j; if UPLO = 'L', BP(i + (j-1)*(2*n-j)/2) = B(i,j) for j<=i<=n.

On exit, the triangular factor U or L from the Cholesky factorization B = U^H *U or B = L*L^H , in the same storage format as B.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the matrix Z of eigenvectors. The eigenvectors are normalized as follows: if ITYPE = 1 or 2, Z^H *B* Z = I; if ITYPE = 3, Z^H *inv(B)* Z = I. If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the required LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If N <= 1, LWORK >= 1. If JOBZ = 'N' and N > 1, LWORK >= N. If JOBZ = 'V' and N > 1, LWORK >= 2*N.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the required sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (MAX(1, LRWORK)). On exit, if INFO = 0, RWORK(1) returns the required LRWORK.

LRWORK Input parameter.

LRWORK is INTEGER

The dimension of array RWORK. If N <= 1, LRWORK >= 1. If JOBZ = 'N' and N > 1, LRWORK >= N. If JOBZ = 'V' and N > 1, LRWORK >= 1 + 5* N + 2*N**2.

If LRWORK = -1, then a workspace query is assumed; the routine only calculates the required sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the required LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of array IWORK. If JOBZ = 'N' or N <= 1, LIWORK >= 1. If JOBZ = 'V' and N > 1, LIWORK >= 3 + 5*N.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the required sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: ZPPTRF or ZHPEVD returned an error code: <= N: if INFO = i, ZHPEVD failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not convergeto zero; > N: if INFO = N + i, for 1 <= i <= n, then the leading minor of order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *chpgvd*. It also exists with a native C interface as *LAPACKE_zhpgvd*.

4.12.51 zhpgvx

zhpgvx computes selected eigenvalues and, optionally, eigenvectors of a complex generalized Hermitian-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x. Here A and B are assumed to be Hermitian, stored in packed format, and B is also positive definite. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhpgvx(ITYPE, JOBZ, RANGE, UPLO, N, AP, BP, VL, VU, IL, IU, ABSTOL,
                  M, W, Z, LDZ, WORK, RWORK, IWORK, IFAIL, INFO)
```

C specification:

```
#include "armpl.h"

void zhpgvx_(const armpl_int_t *itype, const char *jobz, const char *range,
             const char *uplo, const armpl_int_t *n,
             armpl_doublecomplex_t *ap, armpl_doublecomplex_t *bp,
             const double *vl, const double *vu, const armpl_int_t *il,
             const armpl_int_t *iu, const double *abstol, armpl_int_t *m,
             double *w, armpl_doublecomplex_t *z, const armpl_int_t *ldz,
             armpl_doublecomplex_t *work, double *rwork, armpl_int_t *iwork,
             armpl_int_t *ifail, armpl_int_t *info, ... );
```

Parameters

ITYPE Input parameter.

ITYPE is INTEGER

Specifies the problem type to be solved: = 1: A*x = (lambda)*B*x = 2: A*B*x = (lambda)*x = 3: B*A*x = (lambda)*x

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found; = 'V': all eigenvalues in the half-open interval (VL,VU] will be found; = 'I': the IL-th through IU-th eigenvalues will be found.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangles of A and B are stored; = 'L': Lower triangles of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

AP Input and output parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the Hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2*n-j)/2) = A(i,j) for j<=i<=n.

On exit, the contents of AP are destroyed.

BP Input and output parameter.

BP is COMPLEX*16

BP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the Hermitian matrix B, packed columnwise in a linear array. The j-th column of B is stored in the array BP as follows: if UPLO = 'U', BP(i + (j-1)*j/2) = B(i,j) for 1<=i<=j; if UPLO = 'L', BP(i + (j-1)*(2*n-j)/2) = B(i,j) for j<=i<=n.

On exit, the triangular factor U or L from the Cholesky factorization B = U^H *U or B = L*L^H , in the same storage format as B.

VL Input parameter.

VL is DOUBLE PRECISION

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is DOUBLE PRECISION

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is DOUBLE PRECISION

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing AP to tridiagonal form.

Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*DLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*DLAMCH('S').

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). On normal exit, the first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, N). If JOBZ = 'N', then Z is not referenced. If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). The eigenvectors are normalized as follows: if ITYPE = 1 or 2, Z^H *B* Z = I; if ITYPE = 3, Z^H *inv(B)* Z = I.

If an eigenvector fails to converge, then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (7*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (5*N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (N)

If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: ZPPTRF or ZHPEVX returned an error code: <= N: if INFO = i, ZHPEVX failed to converge; i eigenvectors failed to converge. Their indices are stored in array IFAIL. > N: if INFO = N + i, for 1 <= i <= n, then the leading minor of order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *chpgvx*. It also exists with a native C interface as *LAPACKE_zhpgvx*.

4.12.52 zpbstf

zpbstf computes a split Cholesky factorization of a complex Hermitian positive definite band matrix A.

This routine is designed to be used in conjunction with ZHBGST.

The factorization has the form A = S^H *S where S is a band matrix of the same bandwidth as A and the following structure:

```
S = ( U    )
    ( M  L )
```

where U is upper triangular of order m = (n+kd)/2, and L is lower triangular of order n-m.

Syntax

Fortran specification:

```
use armpl_library

subroutine zpbstf(UPLO, N, KD, AB, LDAB, INFO)
```

C specification:

```
#include "armpl.h"

void zpbstf_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
             armpl_doublecomplex_t *ab, const armpl_int_t *ldab,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the Hermitian band matrix A, stored in the first kd+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, if INFO = 0, the factor S from the split Cholesky factorization A = S^H *S. See Further Details.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the factorization could not be completed, because the updated element a(i,i) was negative; the matrix A is not positive definite.

Related Information

For this routine in other precisions, please see *cpbstf*, *dpbstf* and *spbstf*. It also exists with a native C interface as *LAPACKE_zpbstf*.

4.13 LAPACK non symmetric eigenvalues routines

4.13.1 cgebak

cgebak forms the right or left eigenvectors of a complex general matrix by backward transformation on the computed eigenvectors of the balanced matrix output by CGEBAL.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgebak(JOB, SIDE, N, ILO, IHI, SCALE, M, V, LDV, INFO)
```

C specification:

```
#include "armpl.h"

void cgebak_(const char *job, const char *side, const armpl_int_t *n,
             const armpl_int_t *ilo, const armpl_int_t *ihi,
             const float *scale, const armpl_int_t *m,
             armpl_singlecomplex_t *v, const armpl_int_t *ldv,
             armpl_int_t *info, ... );
```

Parameters

JOB Input parameter.

JOB is CHARACTER*1

Specifies the type of backward transformation required: = 'N', do nothing, return immediately; = 'P', do backward transformation for permutation only; = 'S', do backward transformation for scaling only; = 'B', do backward transformations for both permutation and scaling. JOB must be the same as the argument JOB supplied to CGEBAL.

SIDE Input parameter.

SIDE is CHARACTER*1

= 'R': V contains right eigenvectors; = 'L': V contains left eigenvectors.

N Input parameter.

N is INTEGER

The number of rows of the matrix V. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

The integers ILO and IHI determined by CGEBAL. 1 <= ILO <= IHI <= N, if N > 0; ILO=1 and IHI=0, if N=0.

SCALE Input parameter.

SCALE is REAL

SCALE is an array, dimension (N). Details of the permutation and scaling factors, as returned by CGEBAL.

M Input parameter.

M is INTEGER

The number of columns of the matrix V. M >= 0.

V Input and output parameter.

V is COMPLEX

V is an array, dimension (LDV, M). On entry, the matrix of right or left eigenvectors to be transformed, as returned by CHSEIN or CTREVC. On exit, V is overwritten by the transformed eigenvectors.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. LDV >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *dgebak*, *sgebak* and *zgebak*. It also exists with a native C interface as *LAPACKE_cgebak*.

4.13.2 cgebal

cgebal balances a general complex matrix A. This involves, first, permuting A by a similarity transformation to isolate eigenvalues in the first 1 to ILO-1 and last IHI+1 to N elements on the diagonal; and second, applying a diagonal similarity transformation to rows and columns ILO to IHI to make the rows and columns as close in norm as possible. Both steps are optional.

Balancing may reduce the 1-norm of the matrix, and improve the accuracy of the computed eigenvalues and/or eigenvectors.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgebal(JOB, N, A, LDA, ILO, IHI, SCALE, INFO)
```

C specification:

```
#include "armpl.h"

void cgebal_(const char *job, const armpl_int_t *n, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *ilo, armpl_int_t *ihi,
             float *scale, armpl_int_t *info, ... );
```

Parameters

JOB Input parameter.

JOB is CHARACTER*1

Specifies the operations to be performed on A: = 'N': none: simply set ILO = 1, IHI = N, SCALE(I) = 1.0 for i = 1,...,N; = 'P': permute only; = 'S': scale only; = 'B': both permute and scale.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the input matrix A. On exit, A is overwritten by the balanced matrix. If JOB = 'N', A is not referenced. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

ILO Output parameter.

ILO is INTEGER

IHI Output parameter.

IHI is INTEGER

ILO and IHI are set to integers such that on exit A(i,j) = 0 if i > j and j = 1,...,ILO-1 or I = IHI+1,...,N. If JOB = 'N' or 'S', ILO = 1 and IHI = N.

SCALE Output parameter.

SCALE is REAL

SCALE is an array, dimension (N). Details of the permutations and scaling factors applied to A. If P(j) is the index of the row and column interchanged with row and column j and D(j) is the scaling factor applied to row and column j, then SCALE(j) = P(j) for j = 1,...,ILO-1 = D(j) for j = ILO,...,IHI = P(j) for j = IHI+1,...,N. The order in which the interchanges are made is N to IHI+1, then 1 to ILO-1.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *dgebal*, *sgebal* and *zgebal*. It also exists with a native C interface as *LAPACKE_cgebal*.

4.13.3 cgees

cgees computes for an N-by-N complex nonsymmetric matrix A, the eigenvalues, the Schur form T, and, optionally, the matrix of Schur vectors Z. This gives the Schur factorization A = Z*T*(Z^H).

Optionally, it also orders the eigenvalues on the diagonal of the Schur form so that selected eigenvalues are at the top left. The leading columns of Z then form an orthonormal basis for the invariant subspace corresponding to the selected eigenvalues.

A complex matrix is in Schur form if it is upper triangular.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgees(JOBVS, SORT, SELECT, N, A, LDA, SDIM, W, VS, LDVS, WORK,
                 LWORK, RWORK, BWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgees_(const char *jobvs, const char *sort, ARMPL_CGEES_SELECT select,
            const armpl_int_t *n, armpl_singlecomplex_t *a,
            const armpl_int_t *lda, armpl_int_t *sdim,
```

(continues on next page)

(continued from previous page)

```
            armpl_singlecomplex_t *w, armpl_singlecomplex_t *vs,
            const armpl_int_t *ldvs, armpl_singlecomplex_t *work,
            const armpl_int_t *lwork, float *rwork, armpl_int_t *bwork,
            armpl_int_t *info, ... );
```

Parameters

JOBVS Input parameter.

JOBVS is CHARACTER*1

= 'N': Schur vectors are not computed; = 'V': Schur vectors are computed.

SORT Input parameter.

SORT is CHARACTER*1

Specifies whether or not to order the eigenvalues on the diagonal of the Schur form. = 'N': Eigenvalues are not ordered: = 'S': Eigenvalues are ordered (see SELECT).

SELECT Input parameter.

SELECT is a LOGICAL FUNCTION of one COMPLEX argument

SELECT must be declared EXTERNAL in the calling subroutine. If SORT = 'S', SELECT is used to select eigenvalues to order to the top left of the Schur form. IF SORT = 'N', SELECT is not referenced. The eigenvalue W(j) is selected if SELECT(W(j)) is true.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A. On exit, A has been overwritten by its Schur form T.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

SDIM Output parameter.

SDIM is INTEGER

If SORT = 'N', SDIM = 0. If SORT = 'S', SDIM = number of eigenvalues for which SELECT is true.

W Output parameter.

W is COMPLEX

W is an array, dimension (N). W contains the computed eigenvalues, in the same order that they appear on the diagonal of the output Schur form T.

VS Output parameter.

VS is COMPLEX

VS is an array, dimension (LDVS, N). If JOBVS = 'V', VS contains the unitary matrix Z of Schur vectors. If JOBVS = 'N', VS is not referenced.

LDVS Input parameter.

LDVS is INTEGER

The leading dimension of the array VS. LDVS >= 1; if JOBVS = 'V', LDVS >= N.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,2*N). For good performance, LWORK must generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

BWORK Output parameter.

BWORK is LOGICAL

BWORK is an array, dimension (N). Not referenced if SORT = 'N'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, and i is <= N: the QR algorithm failed to compute all the eigenvalues; elements 1:ILO-1 and i+1:N of W contain those eigenvalues which have converged; if JOBVS = 'V', VS contains the matrix which reduces A to its partially converged Schur form. = N+1: the eigenvalues could not be reordered because some eigenvalues were too close to separate (the problem is very ill-conditioned); = N+2: after reordering, roundoff changed values of some complex eigenvalues so that leading eigenvalues in the Schur form no longer satisfy SELECT = .TRUE.. This could also be caused by underflow due to scaling.

Related Information

For this routine in other precisions, please see *dgees*, *sgees* and *zgees*. It also exists with a native C interface as *LAPACKE_cgees*.

4.13.4 cgeesx

cgeesx computes for an N-by-N complex nonsymmetric matrix A, the eigenvalues, the Schur form T, and, optionally, the matrix of Schur vectors Z. This gives the Schur factorization A = Z*T*(Z^H).

Optionally, it also orders the eigenvalues on the diagonal of the Schur form so that selected eigenvalues are at the top left; computes a reciprocal condition number for the average of the selected eigenvalues (RCONDE); and computes a reciprocal condition number for the right invariant subspace corresponding to the selected eigenvalues (RCONDV). The leading columns of Z form an orthonormal basis for this invariant subspace.

For further explanation of the reciprocal condition numbers RCONDE and RCONDV, see Section 4.10 of the LAPACK Users' Guide (where these quantities are called s and sep respectively).

A complex matrix is in Schur form if it is upper triangular.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgeesx(JOBVS, SORT, SELECT, SENSE, N, A, LDA, SDIM, W, VS, LDVS,
                  RCONDE, RCONDV, WORK, LWORK, RWORK, BWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgeesx_(const char *jobvs, const char *sort, ARMPL_CGEESX_SELECT select,
             const char *sense, const armpl_int_t *n,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             armpl_int_t *sdim, armpl_singlecomplex_t *w,
             armpl_singlecomplex_t *vs, const armpl_int_t *ldvs,
             float *rconde, float *rcondv, armpl_singlecomplex_t *work,
             const armpl_int_t *lwork, float *rwork, armpl_int_t *bwork,
             armpl_int_t *info, ... );
```

Parameters

JOBVS Input parameter.

JOBVS is CHARACTER*1

= 'N': Schur vectors are not computed; = 'V': Schur vectors are computed.

SORT Input parameter.

SORT is CHARACTER*1

Specifies whether or not to order the eigenvalues on the diagonal of the Schur form. = 'N': Eigenvalues are not ordered; = 'S': Eigenvalues are ordered (see SELECT).

SELECT Input parameter.

SELECT is a LOGICAL FUNCTION of one COMPLEX argument

SELECT must be declared EXTERNAL in the calling subroutine. If SORT = 'S', SELECT is used to select eigenvalues to order to the top left of the Schur form. If SORT = 'N', SELECT is not referenced. An eigenvalue W(j) is selected if SELECT(W(j)) is true.

SENSE Input parameter.

SENSE is CHARACTER*1

Determines which reciprocal condition numbers are computed. = 'N': None are computed; = 'E': Computed for average of selected eigenvalues only; = 'V': Computed for selected right invariant subspace only; = 'B': Computed for both. If SENSE = 'E', 'V' or 'B', SORT must equal 'S'.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A. On exit, A is overwritten by its Schur form T.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

SDIM Output parameter.

SDIM is INTEGER

If SORT = 'N', SDIM = 0. If SORT = 'S', SDIM = number of eigenvalues for which SELECT is true.

W Output parameter.

W is COMPLEX

W is an array, dimension (N). W contains the computed eigenvalues, in the same order that they appear on the diagonal of the output Schur form T.

VS Output parameter.

VS is COMPLEX

VS is an array, dimension (LDVS, N). If JOBVS = 'V', VS contains the unitary matrix Z of Schur vectors. If JOBVS = 'N', VS is not referenced.

LDVS Input parameter.

LDVS is INTEGER

The leading dimension of the array VS. LDVS >= 1, and if JOBVS = 'V', LDVS >= N.

RCONDE Output parameter.

RCONDE is REAL

If SENSE = 'E' or 'B', RCONDE contains the reciprocal condition number for the average of the selected eigenvalues. Not referenced if SENSE = 'N' or 'V'.

RCONDV Output parameter.

RCONDV is REAL

If SENSE = 'V' or 'B', RCONDV contains the reciprocal condition number for the selected right invariant subspace. Not referenced if SENSE = 'N' or 'E'.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,2*N). Also, if SENSE = 'E' or 'V' or 'B', LWORK >= 2*SDIM*(N-SDIM), where SDIM is the number of selected eigenvalues computed by this routine. Note that 2*SDIM*(N-SDIM) <= N*N/2. Note also that an error is only returned if LWORK < max(1,2*N), but if SENSE = 'E' or 'V' or 'B' this may not be large enough. For good performance, LWORK must generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates upper bound on the optimal size of the array WORK, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

BWORK Output parameter.

BWORK is LOGICAL

BWORK is an array, dimension (N). Not referenced if SORT = 'N'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, and i is <= N: the QR algorithm failed to compute all the eigenvalues; elements 1:ILO-1 and i+1:N of W contain those eigenvalues which have converged; if JOBVS = 'V', VS contains the transformation which reduces A to its partially converged Schur form. = N+1: the eigenvalues could not be reordered because some eigenvalues were too close to separate (the problem is very ill-conditioned); = N+2: after reordering, roundoff changed values of some complex eigenvalues so that leading eigenvalues in the Schur form no longer satisfy SELECT=.TRUE. This could also be caused by underflow due to scaling.

Related Information

For this routine in other precisions, please see *dgeesx*, *sgeesx* and *zgeesx*. It also exists with a native C interface as *LAPACKE_cgeesx*.

4.13.5 cgeev

cgeev computes for an N-by-N complex nonsymmetric matrix A, the eigenvalues and, optionally, the left and/or right eigenvectors.

The right eigenvector v(j) of A satisfies

```
A * v(j) = lambda(j) * v(j)
```

where lambda(j) is its eigenvalue. The left eigenvector u(j) of A satisfies

```
u(j)**H * A = lambda(j) * u(j)**H
```

where $u(j)^H$ denotes the conjugate transpose of u(j).

The computed eigenvectors are normalized to have Euclidean norm equal to 1 and largest component real.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgeev(JOBVL, JOBVR, N, A, LDA, W, VL, LDVL, VR, LDVR, WORK, LWORK,
                 RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgeev_(const char *jobvl, const char *jobvr, const armpl_int_t *n,
            armpl_singlecomplex_t *a, const armpl_int_t *lda,
            armpl_singlecomplex_t *w, armpl_singlecomplex_t *vl,
            const armpl_int_t *ldvl, armpl_singlecomplex_t *vr,
            const armpl_int_t *ldvr, armpl_singlecomplex_t *work,
            const armpl_int_t *lwork, float *rwork, armpl_int_t *info, ... );
```

Parameters

JOBVL Input parameter.

JOBVL is CHARACTER*1

= 'N': left eigenvectors of A are not computed; = 'V': left eigenvectors of are computed.

JOBVR Input parameter.

JOBVR is CHARACTER*1

= 'N': right eigenvectors of A are not computed; = 'V': right eigenvectors of A are computed.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A. On exit, A has been overwritten.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

W Output parameter.

W is COMPLEX

W is an array, dimension (N). W contains the computed eigenvalues.

VL Output parameter.

VL is COMPLEX

VL is an array, dimension (LDVL, N). If JOBVL = 'V', the left eigenvectors u(j) are stored one after another in the columns of VL, in the same order as their eigenvalues. If JOBVL = 'N', VL is not referenced. u(j) = VL(:,j), the j-th column of VL.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the array VL. LDVL >= 1; if JOBVL = 'V', LDVL >= N.

VR Output parameter.

VR is COMPLEX

VR is an array, dimension (LDVR, N). If JOBVR = 'V', the right eigenvectors v(j) are stored one after another in the columns of VR, in the same order as their eigenvalues. If JOBVR = 'N', VR is not referenced. v(j) = VR(:,j), the j-th column of VR.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the array VR. LDVR >= 1; if JOBVR = 'V', LDVR >= N.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,2*N). For good performance, LWORK must generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, the QR algorithm failed to compute all the eigenvalues, and no eigenvectors have been computed; elements and i+1:N of W contain eigenvalues which have converged.

Related Information

For this routine in other precisions, please see *dgeev*, *sgeev* and *zgeev*. It also exists with a native C interface as *LAPACKE_cgeev*.

4.13.6 cgeevx

cgeevx computes for an N-by-N complex nonsymmetric matrix A, the eigenvalues and, optionally, the left and/or right eigenvectors.

Optionally also, it computes a balancing transformation to improve the conditioning of the eigenvalues and eigenvectors (ILO, IHI, SCALE, and ABNRM), reciprocal condition numbers for the eigenvalues (RCONDE), and reciprocal condition numbers for the right eigenvectors (RCONDV).

The right eigenvector v(j) of A satisfies

```
A * v(j) = lambda(j) * v(j)
```

where lambda(j) is its eigenvalue. The left eigenvector u(j) of A satisfies

```
u(j)**H * A = lambda(j) * u(j)**H
```

where $u(j)^H$ denotes the conjugate transpose of u(j).

The computed eigenvectors are normalized to have Euclidean norm equal to 1 and largest component real.

Balancing a matrix means permuting the rows and columns to make it more nearly upper triangular, and applying a diagonal similarity transformation D * A * D**(-1), where D is a diagonal matrix, to make its rows and columns closer in norm and the condition numbers of its eigenvalues and eigenvectors smaller. The computed reciprocal condition numbers correspond to the balanced matrix. Permuting rows and columns will not change the condition numbers (in exact arithmetic) but diagonal scaling will. For further explanation of balancing, see section 4.10.2 of the LAPACK Users' Guide.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgeevx(BALANC, JOBVL, JOBVR, SENSE, N, A, LDA, W, VL, LDVL, VR,
                  LDVR, ILO, IHI, SCALE, ABNRM, RCONDE, RCONDV, WORK, LWORK,
                  RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgeevx_(const char *balanc, const char *jobvl, const char *jobvr,
             const char *sense, const armpl_int_t *n,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             armpl_singlecomplex_t *w, armpl_singlecomplex_t *vl,
             const armpl_int_t *ldvl, armpl_singlecomplex_t *vr,
             const armpl_int_t *ldvr, armpl_int_t *ilo, armpl_int_t *ihi,
             float *scale, float *abnrm, float *rconde, float *rcondv,
             armpl_singlecomplex_t *work, const armpl_int_t *lwork,
             float *rwork, armpl_int_t *info, ... );
```

Parameters

BALANC Input parameter.

BALANC is CHARACTER*1

Indicates how the input matrix should be diagonally scaled and/or permuted to improve the conditioning of its eigenvalues. = 'N': Do not diagonally scale or permute; = 'P': Perform permutations to make the matrix more nearly upper triangular. Do not diagonally scale; = 'S': Diagonally scale the matrix, ie. replace A by D*A*D**(-1), where D is a diagonal matrix chosen to make the rows and columns of A more equal in norm. Do not permute; = 'B': Both diagonally scale and permute A.

Computed reciprocal condition numbers will be for the matrix after balancing and/or permuting. Permuting does not change condition numbers (in exact arithmetic), but balancing does.

JOBVL Input parameter.

JOBVL is CHARACTER*1

= 'N': left eigenvectors of A are not computed; = 'V': left eigenvectors of A are computed. If SENSE = 'E' or 'B', JOBVL must = 'V'.

JOBVR Input parameter.

JOBVR is CHARACTER*1

= 'N': right eigenvectors of A are not computed; = 'V': right eigenvectors of A are computed. If SENSE = 'E' or 'B', JOBVR must = 'V'.

SENSE Input parameter.

SENSE is CHARACTER*1

Determines which reciprocal condition numbers are computed. = 'N': None are computed; = 'E': Computed for eigenvalues only; = 'V': Computed for right eigenvectors only; = 'B': Computed for eigenvalues and right eigenvectors.

If SENSE = 'E' or 'B', both left and right eigenvectors must also be computed (JOBVL = 'V' and JOBVR = 'V').

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A. On exit, A has been overwritten. If JOBVL = 'V' or JOBVR = 'V', A contains the Schur form of the balanced version of the matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

W Output parameter.

W is COMPLEX

W is an array, dimension (N). W contains the computed eigenvalues.

VL Output parameter.

VL is COMPLEX

VL is an array, dimension (LDVL, N). If JOBVL = 'V', the left eigenvectors u(j) are stored one after another in the columns of VL, in the same order as their eigenvalues. If JOBVL = 'N', VL is not referenced. u(j) = VL(:,j), the j-th column of VL.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the array VL. LDVL >= 1; if JOBVL = 'V', LDVL >= N.

VR Output parameter.

VR is COMPLEX

VR is an array, dimension (LDVR, N). If JOBVR = 'V', the right eigenvectors v(j) are stored one after another in the columns of VR, in the same order as their eigenvalues. If JOBVR = 'N', VR is not referenced. v(j) = VR(:,j), the j-th column of VR.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the array VR. LDVR >= 1; if JOBVR = 'V', LDVR >= N.

ILO Output parameter.

ILO is INTEGER

IHI Output parameter.

IHI is INTEGER

ILO and IHI are integer values determined when A was balanced. The balanced A(i,j) = 0 if I > J and J = 1,...,ILO-1 or I = IHI+1,...,N.

SCALE Output parameter.

SCALE is REAL

SCALE is an array, dimension (N). Details of the permutations and scaling factors applied when balancing A. If P(j) is the index of the row and column interchanged with row and column j, and D(j) is the scaling factor applied to row and column j, then SCALE(J) = P(J), for J = 1,...,ILO-1 = D(J), for J = ILO,...,IHI = P(J) for J = IHI+1,...,N. The order in which the interchanges are made is N to IHI+1, then 1 to ILO-1.

ABNRM Output parameter.

ABNRM is REAL

The one-norm of the balanced matrix (the maximum of the sum of absolute values of elements of any column).

RCONDE Output parameter.

RCONDE is REAL

RCONDE is an array, dimension (N). RCONDE(j) is the reciprocal condition number of the j-th eigenvalue.

RCONDV Output parameter.

RCONDV is REAL

RCONDV is an array, dimension (N). RCONDV(j) is the reciprocal condition number of the j-th right eigenvector.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SENSE = 'N' or 'E', LWORK >= max(1,2*N), and if SENSE = 'V' or 'B', LWORK >= N*N+2*N. For good performance, LWORK must generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, the QR algorithm failed to compute all the eigenvalues, and no eigenvectors or condition numbers have been computed; elements 1:ILO-1 and i+1:N of W contain eigenvalues which have converged.

Related Information

For this routine in other precisions, please see *dgeevx*, *sgeevx* and *zgeevx*. It also exists with a native C interface as *LAPACKE_cgeevx*.

4.13.7 cgehrd

cgehrd reduces a complex general matrix A to upper Hessenberg form H by an unitary similarity transformation: $Q^H * A * Q = H$.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgehrd(N, ILO, IHI, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgehrd_(const armpl_int_t *n, const armpl_int_t *ilo,
             const armpl_int_t *ihi, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_singlecomplex_t *tau,
             armpl_singlecomplex_t *work, const armpl_int_t *lwork,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

It is assumed that A is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N. ILO and IHI are normally set by a previous call to CGEBAL; otherwise they should be set to 1 and N respectively. See Further Details. 1 <= ILO <= IHI <= N, if N > 0; ILO=1 and IHI=0, if N=0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the N-by-N general matrix to be reduced. On exit, the upper triangle and the first subdiagonal of A are overwritten with the upper Hessenberg matrix H, and the elements below the first subdiagonal, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TAU Output parameter.

TAU is COMPLEX

TAU is an array, dimension (N-1). The scalar factors of the elementary reflectors (see Further Details). Elements 1:ILO-1 and IHI:N-1 of TAU are set to zero.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (LWORK). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= max(1, N). For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *dgehrd*, *sgehrd* and *zgehrd*. It also exists with a native C interface as *LAPACKE_cgehrd*.

4.13.8 chsein

chsein uses inverse iteration to find specified right and/or left eigenvectors of a complex upper Hessenberg matrix H.

The right eigenvector x and the left eigenvector y of the matrix H corresponding to an eigenvalue w are defined by:

```
H * x = w * x,      y**h * H = w * y**h
```

where y**h denotes the conjugate transpose of the vector y.

Syntax

Fortran specification:

```
use armpl_library

subroutine chsein(SIDE, EIGSRC, INITV, SELECT, N, H, LDH, W, VL, LDVL, VR,
                  LDVR, MM, M, WORK, RWORK, IFAILL, IFAILR, INFO)
```

C specification:

```
#include "armpl.h"

void chsein_(const char *side, const char *eigsrc, const char *initv,
             const armpl_int_t *select, const armpl_int_t *n,
             const armpl_singlecomplex_t *h, const armpl_int_t *ldh,
             armpl_singlecomplex_t *w, armpl_singlecomplex_t *vl,
             const armpl_int_t *ldvl, armpl_singlecomplex_t *vr,
             const armpl_int_t *ldvr, const armpl_int_t *mm, armpl_int_t *m,
             armpl_singlecomplex_t *work, float *rwork, armpl_int_t *ifaill,
             armpl_int_t *ifailr, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'R': compute right eigenvectors only; = 'L': compute left eigenvectors only; = 'B': compute both right and left eigenvectors.

EIGSRC Input parameter.

EIGSRC is CHARACTER*1

Specifies the source of eigenvalues supplied in W: = 'Q': the eigenvalues were found using CHSEQR; thus, if H has zero subdiagonal elements, and so is block-triangular, then the j-th eigenvalue can be assumed to be an eigenvalue of the block containing the j-th row/column. This property allows CHSEIN to perform

inverse iteration on just one diagonal block. = 'N': no assumptions are made on the correspondence between eigenvalues and diagonal blocks. In this case, CHSEIN must always perform inverse iteration using the whole matrix H.

INITV Input parameter.

INITV is CHARACTER*1

= 'N': no initial vectors are supplied; = 'U': user-supplied initial vectors are stored in the arrays VL and/or VR.

SELECT Input parameter.

SELECT is LOGICAL

SELECT is an array, dimension (N). Specifies the eigenvectors to be computed. To select the eigenvector corresponding to the eigenvalue W(j), SELECT(j) must be set to .TRUE..

N Input parameter.

N is INTEGER

The order of the matrix H. N >= 0.

H Input parameter.

H is COMPLEX

H is an array, dimension (LDH, N). The upper Hessenberg matrix H. If a NaN is detected in H, the routine will return with INFO=-6.

LDH Input parameter.

LDH is INTEGER

The leading dimension of the array H. LDH >= max(1, N).

W Input and output parameter.

W is COMPLEX

W is an array, dimension (N). On entry, the eigenvalues of H. On exit, the real parts of W may have been altered since close eigenvalues are perturbed slightly in searching for independent eigenvectors.

VL Input and output parameter.

VL is COMPLEX

VL is an array, dimension (LDVL, MM). On entry, if INITV = 'U' and SIDE = 'L' or 'B', VL must contain starting vectors for the inverse iteration for the left eigenvectors; the starting vector for each eigenvector must be in the same column in which the eigenvector will be stored. On exit, if SIDE = 'L' or 'B', the left eigenvectors specified by SELECT will be stored consecutively in the columns of VL, in the same order as their eigenvalues. If SIDE = 'R', VL is not referenced.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the array VL. LDVL >= max(1, N) if SIDE = 'L' or 'B'; LDVL >= 1 otherwise.

VR Input and output parameter.

VR is COMPLEX

VR is an array, dimension (LDVR, MM). On entry, if INITV = 'U' and SIDE = 'R' or 'B', VR must contain starting vectors for the inverse iteration for the right eigenvectors; the starting vector for each eigenvector must be in the same column in which the eigenvector will be stored. On exit, if SIDE = 'R' or 'B', the right eigenvectors specified by SELECT will be stored consecutively in the columns of VR, in the same order as their eigenvalues. If SIDE = 'L', VR is not referenced.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the array VR. LDVR >= max(1, N) if SIDE = 'R' or 'B'; LDVR >= 1 otherwise.

MM Input parameter.

MM is INTEGER

The number of columns in the arrays VL and/or VR. MM >= M.

M Output parameter.

M is INTEGER

The number of columns in the arrays VL and/or VR required to store the eigenvectors (= the number of .TRUE. elements in SELECT).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (N*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

IFAILL Output parameter.

IFAILL is INTEGER array, dimension (MM)

If SIDE = 'L' or 'B', IFAILL(i) = j > 0 if the left eigenvector in the i-th column of VL (corresponding to the eigenvalue w(j)) failed to converge; IFAILL(i) = 0 if the eigenvector converged satisfactorily. If SIDE = 'R', IFAILL is not referenced.

IFAILR Output parameter.

IFAILR is INTEGER array, dimension (MM)

If SIDE = 'R' or 'B', IFAILR(i) = j > 0 if the right eigenvector in the i-th column of VR (corresponding to the eigenvalue w(j)) failed to converge; IFAILR(i) = 0 if the eigenvector converged satisfactorily. If SIDE = 'L', IFAILR is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, i is the number of eigenvectors which failed to converge; see IFAILL and IFAILR for further details.

Related Information

For this routine in other precisions, please see *dhsein*, *shsein* and *zhsein*. It also exists with a native C interface as *LAPACKE_chsein*.

4.13.9 chseqr

```
CHSEQR computes the eigenvalues of a Hessenberg matrix H
   and, optionally, the matrices T and Z from the Schur decomposition
   H = Z T Z**H, where T is an upper triangular matrix (the
   Schur form), and Z is the unitary matrix of Schur vectors.

   Optionally Z may be postmultiplied into an input unitary
```

(continues on next page)

(continued from previous page)

```
   matrix Q so that this routine can give the Schur factorization
   of a matrix A which has been reduced to the Hessenberg form H
   by the unitary matrix Q:  A = Q*H*Q**H = (QZ)*T*(QZ)**H.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine chseqr(JOB, COMPZ, N, ILO, IHI, H, LDH, W, Z, LDZ, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void chseqr_(const char *job, const char *compz, const armpl_int_t *n,
             const armpl_int_t *ilo, const armpl_int_t *ihi,
             armpl_singlecomplex_t *h, const armpl_int_t *ldh,
             armpl_singlecomplex_t *w, armpl_singlecomplex_t *z,
             const armpl_int_t *ldz, armpl_singlecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

JOB Input parameter.

JOB is CHARACTER*1

= 'E': compute eigenvalues only; = 'S': compute eigenvalues and the Schur form T.

COMPZ Input parameter.

COMPZ is CHARACTER*1

= 'N': no Schur vectors are computed; = 'I': Z is initialized to the unit matrix and the matrix Z of Schur vectors of H is returned; = 'V': Z must contain an unitary matrix Q on entry, and the product Q* Z is returned.

N Input parameter.

N is INTEGER

The order of the matrix H. N .GE. 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

It is assumed that H is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N. ILO and IHI are normally set by a previous call to CGEBAL, and then passed to ZGEHRD when the matrix output by CGEBAL is reduced to Hessenberg form. Otherwise ILO and IHI should be set to 1 and N respectively. If N.GT.0, then 1.LE.ILO.LE.IHI.LE.N. If N = 0, then ILO = 1 and IHI = 0.

H Input and output parameter.

H is COMPLEX

H is an array, dimension (LDH, N). On entry, the upper Hessenberg matrix H. On exit, if INFO = 0 and JOB = 'S', H contains the upper triangular matrix T from the Schur decomposition (the Schur form). If INFO = 0

and JOB = 'E', the contents of H are unspecified on exit. (The output value of H when INFO.GT.0 is given under the description of INFO below.)

Unlike earlier versions of CHSEQR, this subroutine may explicitly H(i,j) = 0 for i.GT.j and j = 1, 2, ... ILO-1 or j = IHI+1, IHI+2, ... N.

LDH Input parameter.

LDH is INTEGER

The leading dimension of the array H. LDH .GE. max(1, N).

W Output parameter.

W is COMPLEX

W is an array, dimension (N). The computed eigenvalues. If JOB = 'S', the eigenvalues are stored in the same order as on the diagonal of the Schur form returned in H, with W(i) = H(i,i).

Z Input and output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, N). If COMPZ = 'N', Z is not referenced. If COMPZ = 'I', on entry Z need not be set and on exit, if INFO = 0, Z contains the unitary matrix Z of the Schur vectors of H. If COMPZ = 'V', on entry Z must contain an N-by-N matrix Q, which is assumed to be equal to the unit matrix except for the submatrix Z(ILO:IHI,ILO:IHI). On exit, if INFO = 0, Z contains Q*Z. Normally Q is the unitary matrix generated by CUNGHR after the call to CGEHRD which formed the Hessenberg matrix H. (The output value of Z when INFO.GT.0 is given under the description of INFO below.)

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. if COMPZ = 'I' or COMPZ = 'V', then LDZ.GE.MAX(1, N). Otherwize, LDZ.GE.1.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (LWORK). On exit, if INFO = 0, WORK(1) returns an estimate of the optimal value for LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK .GE. max(1, N) is sufficient and delivers very good and sometimes optimal performance. However, LWORK as large as 11*N may be required for optimal performance. A workspace query is recommended to determine the optimal workspace size.

If LWORK = -1, then CHSEQR does a workspace query. In this case, CHSEQR checks the input parameters and estimates the optimal workspace size for the given values of N, ILO and IHI. The estimate is returned in WORK(1). No error message related to LWORK is issued by XERBLA. Neither H nor Z are accessed.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit .LT. 0: if INFO = -i, the i-th argument had an illegal value .GT. 0: if INFO = i, CHSEQR failed to compute all of the eigenvalues. Elements 1:ilo-1 and i+1:n of WR and WI contain those eigenvalues which have been successfully computed. (Failures are rare.)

If INFO .GT. 0 and JOB = 'E', then on exit, the remaining unconverged eigenvalues are the eigen- values of the upper Hessenberg matrix rows and columns ILO through INFO of the final, output value of H.

If INFO .GT. 0 and JOB = 'S', then on exit

(*) (initial value of H)*U = U*(final value of H)

where U is a unitary matrix. The final value of `H` is upper Hessenberg and triangular in rows and columns INFO+1 through `IHI`.

If `INFO` .GT. 0 and `COMPZ` = 'V', then on exit

(final value of Z) = (initial value of Z)*U

where U is the unitary matrix in (*) (regard- less of the value of `JOB`.)

If `INFO` .GT. 0 and `COMPZ` = 'I', then on exit (final value of Z) = U where U is the unitary matrix in (*) (regard- less of the value of `JOB`.)

If `INFO` .GT. 0 and `COMPZ` = 'N', then `Z` is not accessed.

Related Information

For this routine in other precisions, please see *dhseqr*, *shseqr* and *zhseqr*. It also exists with a native C interface as *LAPACKE_chseqr*.

4.13.10 ctrevc

`ctrevc` computes some or all of the right and/or left eigenvectors of a complex upper triangular matrix T. Matrices of this type are produced by the Schur factorization of a complex general matrix: A = Q*T*Q^H , as computed by CHSEQR.

The right eigenvector x and the left eigenvector y of T corresponding to an eigenvalue w are defined by:

```
T*x = w*x,      (y**H)*T = w*(y**H)
```

where y^H denotes the conjugate transpose of the vector y. The eigenvalues are not input to this routine, but are read directly from the diagonal of T.

This routine returns the matrices X and/or Y of right and left eigenvectors of T, or the products Q*X and/or Q*Y, where Q is an input matrix. If Q is the unitary factor that reduces a matrix A to Schur form T, then Q*X and Q*Y are the matrices of right and left eigenvectors of A.

Syntax

Fortran specification:

```
use armpl_library

subroutine ctrevc(SIDE, HOWMNY, SELECT, N, T, LDT, VL, LDVL, VR, LDVR, MM, M,
                  WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ctrevc_(const char *side, const char *howmny, const armpl_int_t *select,
             const armpl_int_t *n, armpl_singlecomplex_t *t,
             const armpl_int_t *ldt, armpl_singlecomplex_t *vl,
             const armpl_int_t *ldvl, armpl_singlecomplex_t *vr,
             const armpl_int_t *ldvr, const armpl_int_t *mm, armpl_int_t *m,
             armpl_singlecomplex_t *work, float *rwork, armpl_int_t *info,
             ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'R': compute right eigenvectors only; = 'L': compute left eigenvectors only; = 'B': compute both right and left eigenvectors.

HOWMNY Input parameter.

HOWMNY is CHARACTER*1

= 'A': compute all right and/or left eigenvectors; = 'B': compute all right and/or left eigenvectors, backtransformed using the matrices supplied in VR and/or VL; = 'S': compute selected right and/or left eigenvectors, as indicated by the logical array SELECT.

SELECT Input parameter.

SELECT is LOGICAL

SELECT is an array, dimension (N). If HOWMNY = 'S', SELECT specifies the eigenvectors to be computed. The eigenvector corresponding to the j-th eigenvalue is computed if SELECT(j) = .TRUE.. Not referenced if HOWMNY = 'A' or 'B'.

N Input parameter.

N is INTEGER

The order of the matrix T. N >= 0.

T Input and output parameter.

T is COMPLEX

T is an array, dimension (LDT, N). The upper triangular matrix T. T is modified, but restored on exit.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

VL Input and output parameter.

VL is COMPLEX

VL is an array, dimension (LDVL, MM). On entry, if SIDE = 'L' or 'B' and HOWMNY = 'B', VL must contain an N-by-N matrix Q (usually the unitary matrix Q of Schur vectors returned by CHSEQR). On exit, if SIDE = 'L' or 'B', VL contains: if HOWMNY = 'A', the matrix Y of left eigenvectors of T; if HOWMNY = 'B', the matrix Q*Y; if HOWMNY = 'S', the left eigenvectors of T specified by SELECT, stored consecutively in the columns of VL, in the same order as their eigenvalues. Not referenced if SIDE = 'R'.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the array VL. LDVL >= 1, and if SIDE = 'L' or 'B', LDVL >= N.

VR Input and output parameter.

VR is COMPLEX

VR is an array, dimension (LDVR, MM). On entry, if SIDE = 'R' or 'B' and HOWMNY = 'B', VR must contain an N-by-N matrix Q (usually the unitary matrix Q of Schur vectors returned by CHSEQR). On exit, if SIDE = 'R' or 'B', VR contains: if HOWMNY = 'A', the matrix X of right eigenvectors of T; if HOWMNY = 'B', the matrix Q*X; if HOWMNY = 'S', the right eigenvectors of T specified by SELECT, stored consecutively in the columns of VR, in the same order as their eigenvalues. Not referenced if SIDE = 'L'.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the array VR. LDVR >= 1, and if SIDE = 'R' or 'B'; LDVR >= N.

MM Input parameter.

MM is INTEGER

The number of columns in the arrays VL and/or VR. MM >= M.

M Output parameter.

M is INTEGER

The number of columns in the arrays VL and/or VR actually used to store the eigenvectors. If HOWMNY = 'A' or 'B', M is set to N. Each selected eigenvector occupies one column.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dtrevc*, *strevc* and *ztrevc*. It also exists with a native C interface as *LAPACKE_ctrevc*.

4.13.11 ctrexc

ctrexc reorders the Schur factorization of a complex matrix A = Q*T*Q^H , so that the diagonal element of T with row index IFST is moved to row ILST.

The Schur form T is reordered by a unitary similarity transformation Z^H *T*Z, and optionally the matrix Q of Schur vectors is updated by postmultplying it with Z.

Syntax

Fortran specification:

```
use armpl_library

subroutine ctrexc(COMPQ, N, T, LDT, Q, LDQ, IFST, ILST, INFO)
```

C specification:

```
#include "armpl.h"

void ctrexc_(const char *compq, const armpl_int_t *n,
             armpl_singlecomplex_t *t, const armpl_int_t *ldt,
             armpl_singlecomplex_t *q, const armpl_int_t *ldq,
```

(continues on next page)

(continued from previous page)

```
            const armpl_int_t *ifst, const armpl_int_t *ilst,
            armpl_int_t *info, ... );
```

Parameters

COMPQ Input parameter.

COMPQ is CHARACTER*1

= 'V': update the matrix Q of Schur vectors; = 'N': do not update Q.

N Input parameter.

N is INTEGER

The order of the matrix T. N >= 0. If N == 0 arguments ILST and IFST may be any value.

T Input and output parameter.

T is COMPLEX

T is an array, dimension (LDT, N). On entry, the upper triangular matrix T. On exit, the reordered upper triangular matrix.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

Q Input and output parameter.

Q is COMPLEX

Q is an array, dimension (LDQ, N). On entry, if COMPQ = 'V', the matrix Q of Schur vectors. On exit, if COMPQ = 'V', Q has been postmultiplied by the unitary transformation matrix Z which reorders T. If COMPQ = 'N', Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= 1, and if COMPQ = 'V', LDQ >= max(1, N).

IFST Input parameter.

IFST is INTEGER

ILST Input parameter.

ILST is INTEGER

Specify the reordering of the diagonal elements of T: The element with row index IFST is moved to row ILST by a sequence of transpositions between adjacent elements. 1 <= IFST <= N; 1 <= ILST <= N.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dtrexc*, *strexc* and *ztrexc*. It also exists with a native C interface as *LAPACKE_ctrexc*.

4.13.12 ctrsen

`ctrsen` reorders the Schur factorization of a complex matrix A = Q*T*Q^H , so that a selected cluster of eigenvalues appears in the leading positions on the diagonal of the upper triangular matrix T, and the leading columns of Q form an orthonormal basis of the corresponding right invariant subspace.

Optionally the routine computes the reciprocal condition numbers of the cluster of eigenvalues and/or the invariant subspace.

Syntax

Fortran specification:

```
use armpl_library

subroutine ctrsen(JOB, COMPQ, SELECT, N, T, LDT, Q, LDQ, W, M, S, SEP, WORK,
                  LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ctrsen_(const char *job, const char *compq, const armpl_int_t *select,
             const armpl_int_t *n, armpl_singlecomplex_t *t,
             const armpl_int_t *ldt, armpl_singlecomplex_t *q,
             const armpl_int_t *ldq, armpl_singlecomplex_t *w, armpl_int_t *m,
             float *s, float *sep, armpl_singlecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

JOB Input parameter.

JOB is CHARACTER*1

Specifies whether condition numbers are required for the cluster of eigenvalues (S) or the invariant subspace (SEP): = 'N': none; = 'E': for eigenvalues only (S); = 'V': for invariant subspace only (SEP); = 'B': for both eigenvalues and invariant subspace (S and SEP).

COMPQ Input parameter.

COMPQ is CHARACTER*1

= 'V': update the matrix Q of Schur vectors; = 'N': do not update Q.

SELECT Input parameter.

SELECT is LOGICAL

SELECT is an array, dimension (N). SELECT specifies the eigenvalues in the selected cluster. To select the j-th eigenvalue, SELECT(j) must be set to .TRUE..

N Input parameter.

N is INTEGER

The order of the matrix T. N >= 0.

T Input and output parameter.

T is COMPLEX

T is an array, dimension (LDT, N). On entry, the upper triangular matrix T. On exit, T is overwritten by the reordered matrix T, with the selected eigenvalues as the leading diagonal elements.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

Q Input and output parameter.

Q is COMPLEX

Q is an array, dimension (LDQ, N). On entry, if COMPQ = 'V', the matrix Q of Schur vectors. On exit, if COMPQ = 'V', Q has been postmultiplied by the unitary transformation matrix which reorders T; the leading M columns of Q form an orthonormal basis for the specified invariant subspace. If COMPQ = 'N', Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= 1; and if COMPQ = 'V', LDQ >= N.

W Output parameter.

W is COMPLEX

W is an array, dimension (N). The reordered eigenvalues of T, in the same order as they appear on the diagonal of T.

M Output parameter.

M is INTEGER

The dimension of the specified invariant subspace. 0 <= M <= N.

S Output parameter.

S is REAL

If JOB = 'E' or 'B', S is a lower bound on the reciprocal condition number for the selected cluster of eigenvalues. S cannot underestimate the true reciprocal condition number by more than a factor of sqrt(N). If M = 0 or N, S = 1. If JOB = 'N' or 'V', S is not referenced.

SEP Output parameter.

SEP is REAL

If JOB = 'V' or 'B', SEP is the estimated reciprocal condition number of the specified invariant subspace. If M = 0 or N, SEP = norm(T). If JOB = 'N' or 'E', SEP is not referenced.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If JOB = 'N', LWORK >= 1; if JOB = 'E', LWORK = max(1,M*(N-M)); if JOB = 'V' or 'B', LWORK >= max(1,2*M*(N-M)).

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dtrsen*, *strsen* and *ztrsen*. It also exists with a native C interface as *LAPACKE_ctrsen*.

4.13.13 ctrsna

`ctrsna` estimates reciprocal condition numbers for specified eigenvalues and/or right eigenvectors of a complex upper triangular matrix T (or of any matrix Q*T*Q^H with Q unitary).

Syntax

Fortran specification:

```
use armpl_library

subroutine ctrsna(JOB, HOWMNY, SELECT, N, T, LDT, VL, LDVL, VR, LDVR, S, SEP,
                  MM, M, WORK, LDWORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ctrsna_(const char *job, const char *howmny, const armpl_int_t *select,
             const armpl_int_t *n, const armpl_singlecomplex_t *t,
             const armpl_int_t *ldt, const armpl_singlecomplex_t *vl,
             const armpl_int_t *ldvl, const armpl_singlecomplex_t *vr,
             const armpl_int_t *ldvr, float *s, float *sep,
             const armpl_int_t *mm, armpl_int_t *m,
             armpl_singlecomplex_t *work, const armpl_int_t *ldwork,
             float *rwork, armpl_int_t *info, ... );
```

Parameters

JOB Input parameter.

`JOB` is `CHARACTER*1`

Specifies whether condition numbers are required for eigenvalues (`S`) or eigenvectors (`SEP`): = 'E': for eigenvalues only (`S`); = 'V': for eigenvectors only (`SEP`); = 'B': for both eigenvalues and eigenvectors (S and SEP).

HOWMNY Input parameter.

`HOWMNY` is `CHARACTER*1`

= 'A': compute condition numbers for all eigenpairs; = 'S': compute condition numbers for selected eigenpairs specified by the array `SELECT`.

SELECT Input parameter.

`SELECT` is `LOGICAL`

`SELECT` is an array, dimension (`N`). If `HOWMNY` = 'S', `SELECT` specifies the eigenpairs for which condition numbers are required. To select condition numbers for the j-th eigenpair, SELECT(j) must be set to .TRUE.. If `HOWMNY` = 'A', `SELECT` is not referenced.

N Input parameter.

`N` is `INTEGER`

The order of the matrix `T`. `N` >= 0.

T Input parameter.

T is COMPLEX

T is an array, dimension (LDT, N). The upper triangular matrix T.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

VL Input parameter.

VL is COMPLEX

VL is an array, dimension (LDVL, M). If JOB = 'E' or 'B', VL must contain left eigenvectors of T (or of any Q*T*Q^H with Q unitary), corresponding to the eigenpairs specified by HOWMNY and SELECT. The eigenvectors must be stored in consecutive columns of VL, as returned by CHSEIN or CTREVC. If JOB = 'V', VL is not referenced.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the array VL. LDVL >= 1; and if JOB = 'E' or 'B', LDVL >= N.

VR Input parameter.

VR is COMPLEX

VR is an array, dimension (LDVR, M). If JOB = 'E' or 'B', VR must contain right eigenvectors of T (or of any Q*T*Q^H with Q unitary), corresponding to the eigenpairs specified by HOWMNY and SELECT. The eigenvectors must be stored in consecutive columns of VR, as returned by CHSEIN or CTREVC. If JOB = 'V', VR is not referenced.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the array VR. LDVR >= 1; and if JOB = 'E' or 'B', LDVR >= N.

S Output parameter.

S is REAL

S is an array, dimension (MM). If JOB = 'E' or 'B', the reciprocal condition numbers of the selected eigenvalues, stored in consecutive elements of the array. Thus S(j), SEP(j), and the j-th columns of VL and VR all correspond to the same eigenpair (but not in general the j-th eigenpair, unless all eigenpairs are selected). If JOB = 'V', S is not referenced.

SEP Output parameter.

SEP is REAL

SEP is an array, dimension (MM). If JOB = 'V' or 'B', the estimated reciprocal condition numbers of the selected eigenvectors, stored in consecutive elements of the array. If JOB = 'E', SEP is not referenced.

MM Input parameter.

MM is INTEGER

The number of elements in the arrays S (if JOB = 'E' or 'B') and/or SEP (if JOB = 'V' or 'B'). MM >= M.

M Output parameter.

M is INTEGER

The number of elements of the arrays S and/or SEP actually used to store the estimated condition numbers. If HOWMNY = 'A', M is set to N.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (LDWORK,N+6). If JOB = 'E', WORK is not referenced.

LDWORK Input parameter.

LDWORK is INTEGER

The leading dimension of the array WORK. LDWORK >= 1; and if JOB = 'V' or 'B', LDWORK >= N.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N). If JOB = 'E', RWORK is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dtrsna*, *strsna* and *ztrsna*. It also exists with a native C interface as *LAPACKE_ctrsna*.

4.13.14 ctrsyl

ctrsyl solves the complex Sylvester matrix equation:

```
op(A)*X + X*op(B) = scale*C or
op(A)*X - X*op(B) = scale*C,
```

where op(A) = A or A^H , and A and B are both upper triangular. A is M-by-M and B is N-by-N; the right hand side C and the solution X are M-by-N; and scale is an output scale factor, set <= 1 to avoid overflow in X.

Syntax

Fortran specification:

```
use armpl_library

subroutine ctrsyl(TRANA, TRANB, ISGN, M, N, A, LDA, B, LDB, C, LDC, SCALE,
                  INFO)
```

C specification:

```
#include "armpl.h"

void ctrsyl_(const char *trana, const char *tranb, const armpl_int_t *isgn,
             const armpl_int_t *m, const armpl_int_t *n,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const armpl_singlecomplex_t *b, const armpl_int_t *ldb,
             armpl_singlecomplex_t *c, const armpl_int_t *ldc, float *scale,
             armpl_int_t *info, ... );
```

Parameters

TRANA Input parameter.

TRANA is CHARACTER*1

Specifies the option op(A): = 'N': op(A) = A (No transpose) = 'C': op(A) = A^H (Conjugate transpose)

TRANB Input parameter.

TRANB is CHARACTER*1

Specifies the option op(B): = 'N': op(B) = B (No transpose) = 'C': op(B) = B^H (Conjugate transpose)

ISGN Input parameter.

ISGN is INTEGER

Specifies the sign in the equation: = +1: solve op(A)*X + X*op(B) = scale*C = -1: solve op(A)*X - X*op(B) = scale* C

M Input parameter.

M is INTEGER

The order of the matrix A, and the number of rows in the matrices X and C. M >= 0.

N Input parameter.

N is INTEGER

The order of the matrix B, and the number of columns in the matrices X and C. N >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, M). The upper triangular matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, N). The upper triangular matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). On entry, the M-by-N right hand side matrix C. On exit, C is overwritten by the solution matrix X.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M)

SCALE Output parameter.

SCALE is REAL

The scale factor, scale, set <= 1 to avoid overflow in X.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value = 1: A and B have common or very close eigenvalues; perturbed values were used to solve the equation (but the matrices A and B are unchanged).

Related Information

For this routine in other precisions, please see *dtrsyl*, *strsyl* and *ztrsyl*. It also exists with a native C interface as *LAPACKE_ctrsyl*.

4.13.15 cunghr

cunghr generates a complex unitary matrix Q which is defined as the product of IHI-ILO elementary reflectors of order N, as returned by CGEHRD:

Q = H(ilo) H(ilo+1) . . . H(ihi-1).

Syntax

Fortran specification:

```
use armpl_library

subroutine cunghr(N, ILO, IHI, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cunghr_(const armpl_int_t *n, const armpl_int_t *ilo,
             const armpl_int_t *ihi, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, const armpl_singlecomplex_t *tau,
             armpl_singlecomplex_t *work, const armpl_int_t *lwork,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix Q. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

ILO and IHI must have the same values as in the previous call of CGEHRD. Q is equal to the unit matrix except in the submatrix Q(ilo+1:ihi,ilo+1:ihi). 1 <= ILO <= IHI <= N, if N > 0; ILO=1 and IHI=0, if N=0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the vectors which define the elementary reflectors, as returned by CGEHRD. On exit, the N-by-N unitary matrix Q.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TAU Input parameter.

TAU is COMPLEX

TAU is an array, dimension (N-1). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by CGEHRD.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= IHI-ILO. For optimum performance LWORK >= (IHI-ILO)*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zunghr*. It also exists with a native C interface as *LAPACKE_cunghr*.

4.13.16 cunmhr

cunmhr overwrites the general complex M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'C': Q^H * C C * Q^H

where Q is a complex unitary matrix of order nq, with nq = m if SIDE = 'L' and nq = n if SIDE = 'R'. Q is defined as the product of IHI-ILO elementary reflectors, as returned by CGEHRD:

Q = H(ilo) H(ilo+1) . . . H(ihi-1).

Syntax

Fortran specification:

```
use armpl_library

subroutine cunmhr(SIDE, TRANS, M, N, ILO, IHI, A, LDA, TAU, C, LDC, WORK,
                  LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cunmhr_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *ilo,
             const armpl_int_t *ihi, const armpl_singlecomplex_t *a,
             const armpl_int_t *lda, const armpl_singlecomplex_t *tau,
             armpl_singlecomplex_t *c, const armpl_int_t *ldc,
             armpl_singlecomplex_t *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left; = 'R': apply Q or Q^H from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply Q (No transpose) = 'C': apply Q^H (Conjugate transpose)

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

ILO and IHI must have the same values as in the previous call of CGEHRD. Q is equal to the unit matrix except in the submatrix Q(ilo+1:ihi,ilo+1:ihi). If SIDE = 'L', then 1 <= ILO <= IHI <= M, if M > 0, and ILO = 1 and IHI = 0, if M = 0; if SIDE = 'R', then 1 <= ILO <= IHI <= N, if N > 0, and ILO = 1 and IHI = 0, if N = 0.

A Input parameter.

A is COMPLEX

A is an array, dimension. (LDA, M) if SIDE = 'L' (LDA, N) if SIDE = 'R' The vectors which define the elementary reflectors, as returned by CGEHRD.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M) if SIDE = 'L'; LDA >= max(1, N) if SIDE = 'R'.

TAU Input parameter.

TAU is COMPLEX

TAU is an array, dimension. (M-1) if SIDE = 'L' (N-1) if SIDE = 'R' TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by CGEHRD.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^H * C or C*Q^H or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, N); if SIDE = 'R', LWORK >= max(1, M). For optimum performance LWORK >= N*NB if SIDE = 'L', and LWORK >= M*NB if SIDE = 'R', where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zunmhr*. It also exists with a native C interface as *LAPACKE_cunmhr*.

4.13.17 dgebak

dgebak forms the right or left eigenvectors of a real general matrix by backward transformation on the computed eigenvectors of the balanced matrix output by DGEBAL.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgebak(JOB, SIDE, N, ILO, IHI, SCALE, M, V, LDV, INFO)
```

C specification:

```
#include "armpl.h"

void dgebak_(const char *job, const char *side, const armpl_int_t *n,
             const armpl_int_t *ilo, const armpl_int_t *ihi,
             const double *scale, const armpl_int_t *m, double *v,
             const armpl_int_t *ldv, armpl_int_t *info, ... );
```

Parameters

JOB Input parameter.

JOB is CHARACTER*1

Specifies the type of backward transformation required: = 'N', do nothing, return immediately; = 'P', do backward transformation for permutation only; = 'S', do backward transformation for scaling only; = 'B', do backward transformations for both permutation and scaling. JOB must be the same as the argument JOB supplied to DGEBAL.

SIDE Input parameter.

SIDE is CHARACTER*1

= 'R': V contains right eigenvectors; = 'L': V contains left eigenvectors.

N Input parameter.

N is INTEGER

The number of rows of the matrix V. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

The integers ILO and IHI determined by DGEBAL. 1 <= ILO <= IHI <= N, if N > 0; ILO=1 and IHI=0, if N=0.

SCALE Input parameter.

SCALE is DOUBLE PRECISION

SCALE is an array, dimension (N). Details of the permutation and scaling factors, as returned by DGEBAL.

M Input parameter.

M is INTEGER

The number of columns of the matrix V. M >= 0.

V Input and output parameter.

V is DOUBLE PRECISION

V is an array, dimension (LDV, M). On entry, the matrix of right or left eigenvectors to be transformed, as returned by DHSEIN or DTREVC. On exit, V is overwritten by the transformed eigenvectors.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. LDV >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cgebak*, *sgebak* and *zgebak*. It also exists with a native C interface as *LAPACKE_dgebak*.

4.13.18 dgebal

`dgebal` balances a general real matrix A. This involves, first, permuting A by a similarity transformation to isolate eigenvalues in the first 1 to ILO-1 and last IHI+1 to N elements on the diagonal; and second, applying a diagonal similarity transformation to rows and columns ILO to IHI to make the rows and columns as close in norm as possible. Both steps are optional.

Balancing may reduce the 1-norm of the matrix, and improve the accuracy of the computed eigenvalues and/or eigenvectors.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgebal(JOB, N, A, LDA, ILO, IHI, SCALE, INFO)
```

C specification:

```
#include "armpl.h"

void dgebal_(const char *job, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, armpl_int_t *ilo, armpl_int_t *ihi,
             double *scale, armpl_int_t *info, ... );
```

Parameters

JOB Input parameter.

`JOB` is `CHARACTER*1`

Specifies the operations to be performed on A: = 'N': none: simply set `ILO` = 1, `IHI` = `N`, SCALE(I) = 1.0 for i = 1,...,N; = 'P': permute only; = 'S': scale only; = 'B': both permute and scale.

N Input parameter.

`N` is `INTEGER`

The order of the matrix `A`. `N` >= 0.

A Input and output parameter.

`A` is `DOUBLE PRECISION`

`A` is an array, dimension (`LDA`, `N`). On entry, the input matrix `A`. On exit, `A` is overwritten by the balanced matrix. If `JOB` = 'N', `A` is not referenced. See Further Details.

LDA Input parameter.

`LDA` is `INTEGER`

The leading dimension of the array `A`. `LDA` >= max(1, `N`).

ILO Output parameter.

`ILO` is `INTEGER`

IHI Output parameter.

`IHI` is `INTEGER`

`ILO` and `IHI` are set to integers such that on exit A(i,j) = 0 if i > j and j = 1,...,ILO-1 or I = IHI+1,...,N. If `JOB` = 'N' or 'S', `ILO` = 1 and `IHI` = `N`.

SCALE Output parameter.

SCALE is DOUBLE PRECISION

SCALE is an array, dimension (N). Details of the permutations and scaling factors applied to A. If P(j) is the index of the row and column interchanged with row and column j and D(j) is the scaling factor applied to row and column j, then SCALE(j) = P(j) for j = 1,...,ILO-1 = D(j) for j = ILO,...,IHI = P(j) for j = IHI+1,...,N. The order in which the interchanges are made is N to IHI+1, then 1 to ILO-1.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cgebal*, *sgebal* and *zgebal*. It also exists with a native C interface as *LAPACKE_dgebal*.

4.13.19 dgees

dgees computes for an N-by-N real nonsymmetric matrix A, the eigenvalues, the real Schur form T, and, optionally, the matrix of Schur vectors Z. This gives the Schur factorization A = Z*T*(Z^T).

Optionally, it also orders the eigenvalues on the diagonal of the real Schur form so that selected eigenvalues are at the top left. The leading columns of Z then form an orthonormal basis for the invariant subspace corresponding to the selected eigenvalues.

A matrix is in real Schur form if it is upper quasi-triangular with 1-by-1 and 2-by-2 blocks. 2-by-2 blocks will be standardized in the form

```
[  a  b  ]
[  c  a  ]
```

where b*c < 0. The eigenvalues of such a block are a +- sqrt(bc).

Syntax

Fortran specification:

```
use armpl_library

subroutine dgees(JOBVS, SORT, SELECT, N, A, LDA, SDIM, WR, WI, VS, LDVS, WORK,
                 LWORK, BWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgees_(const char *jobvs, const char *sort, ARMPL_DGEES_SELECT select,
            const armpl_int_t *n, double *a, const armpl_int_t *lda,
            armpl_int_t *sdim, double *wr, double *wi, double *vs,
            const armpl_int_t *ldvs, double *work, const armpl_int_t *lwork,
            armpl_int_t *bwork, armpl_int_t *info, ... );
```

Parameters

JOBVS Input parameter.

JOBVS is CHARACTER*1

= 'N': Schur vectors are not computed; = 'V': Schur vectors are computed.

SORT Input parameter.

SORT is CHARACTER*1

Specifies whether or not to order the eigenvalues on the diagonal of the Schur form. = 'N': Eigenvalues are not ordered; = 'S': Eigenvalues are ordered (see SELECT).

SELECT Input parameter.

SELECT is a LOGICAL FUNCTION of two DOUBLE PRECISION arguments

SELECT must be declared EXTERNAL in the calling subroutine. If SORT = 'S', SELECT is used to select eigenvalues to sort to the top left of the Schur form. If SORT = 'N', SELECT is not referenced. An eigenvalue WR(j)+sqrt(-1)*WI(j) is selected if SELECT(WR(j),WI(j)) is true; i.e., if either one of a complex conjugate pair of eigenvalues is selected, then both complex eigenvalues are selected. Note that a selected complex eigenvalue may no longer satisfy SELECT(WR(j),WI(j)) = .TRUE. after ordering, since ordering may change the value of complex eigenvalues (especially if the eigenvalue is ill-conditioned); in this case INFO is set to N+2 (see INFO below).

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A. On exit, A has been overwritten by its real Schur form T.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

SDIM Output parameter.

SDIM is INTEGER

If SORT = 'N', SDIM = 0. If SORT = 'S', SDIM = number of eigenvalues (after sorting) for which SELECT is true. (Complex conjugate pairs for which SELECT is true for either eigenvalue count as 2.)

WR Output parameter.

WR is DOUBLE PRECISION

WR is an array, dimension (N) .

WI Output parameter.

WI is DOUBLE PRECISION

WI is an array, dimension (N). WR and WI contain the real and imaginary parts, respectively, of the computed eigenvalues in the same order that they appear on the diagonal of the output Schur form T. Complex conjugate pairs of eigenvalues will appear consecutively with the eigenvalue having the positive imaginary part first.

VS Output parameter.

VS is DOUBLE PRECISION

VS is an array, dimension (LDVS, N). If JOBVS = 'V', VS contains the orthogonal matrix Z of Schur vectors. If JOBVS = 'N', VS is not referenced.

LDVS Input parameter.

LDVS is INTEGER

The leading dimension of the array VS. LDVS >= 1; if JOBVS = 'V', LDVS >= N.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) contains the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,3*N). For good performance, LWORK must generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

BWORK Output parameter.

BWORK is LOGICAL

BWORK is an array, dimension (N). Not referenced if SORT = 'N'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, and i is <= N: the QR algorithm failed to compute all the eigenvalues; elements 1:ILO-1 and i+1:N of WR and WI contain those eigenvalues which have converged; if JOBVS = 'V', VS contains the matrix which reduces A to its partially converged Schur form. = N+1: the eigenvalues could not be reordered because some eigenvalues were too close to separate (the problem is very ill-conditioned); = N+2: after reordering, roundoff changed values of some complex eigenvalues so that leading eigenvalues in the Schur form no longer satisfy SELECT=.TRUE. This could also be caused by underflow due to scaling.

Related Information

For this routine in other precisions, please see *cgees*, *sgees* and *zgees*. It also exists with a native C interface as *LAPACKE_dgees*.

4.13.20 dgeesx

dgeesx computes for an N-by-N real nonsymmetric matrix A, the eigenvalues, the real Schur form T, and, optionally, the matrix of Schur vectors Z. This gives the Schur factorization A = Z*T*(Z^T).

Optionally, it also orders the eigenvalues on the diagonal of the real Schur form so that selected eigenvalues are at the top left; computes a reciprocal condition number for the average of the selected eigenvalues (RCONDE); and computes a reciprocal condition number for the right invariant subspace corresponding to the selected eigenvalues (RCONDV). The leading columns of Z form an orthonormal basis for this invariant subspace.

For further explanation of the reciprocal condition numbers RCONDE and RCONDV, see Section 4.10 of the LAPACK Users' Guide (where these quantities are called s and sep respectively).

A real matrix is in real Schur form if it is upper quasi-triangular with 1-by-1 and 2-by-2 blocks. 2-by-2 blocks will be standardized in the form

```
[  a  b  ]
[  c  a  ]
```

where b*c < 0. The eigenvalues of such a block are a +- sqrt(bc).

Syntax

Fortran specification:

```
use armpl_library

subroutine dgeesx(JOBVS, SORT, SELECT, SENSE, N, A, LDA, SDIM, WR, WI, VS,
                  LDVS, RCONDE, RCONDV, WORK, LWORK, IWORK, LIWORK, BWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dgeesx_(const char *jobvs, const char *sort, ARMPL_DGEESX_SELECT select,
             const char *sense, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, armpl_int_t *sdim, double *wr,
             double *wi, double *vs, const armpl_int_t *ldvs, double *rconde,
             double *rcondv, double *work, const armpl_int_t *lwork,
             armpl_int_t *iwork, const armpl_int_t *liwork,
             armpl_int_t *bwork, armpl_int_t *info, ... );
```

Parameters

JOBVS Input parameter.

JOBVS is CHARACTER*1

= 'N': Schur vectors are not computed; = 'V': Schur vectors are computed.

SORT Input parameter.

SORT is CHARACTER*1

Specifies whether or not to order the eigenvalues on the diagonal of the Schur form. = 'N': Eigenvalues are not ordered; = 'S': Eigenvalues are ordered (see SELECT).

SELECT Input parameter.

SELECT is a LOGICAL FUNCTION of two DOUBLE PRECISION arguments

SELECT must be declared EXTERNAL in the calling subroutine. If SORT = 'S', SELECT is used to select eigenvalues to sort to the top left of the Schur form. If SORT = 'N', SELECT is not referenced. An eigenvalue WR(j)+sqrt(-1)*WI(j) is selected if SELECT(WR(j),WI(j)) is true; i.e., if either one of a complex conjugate pair of eigenvalues is selected, then both are. Note that a selected complex eigenvalue may no longer satisfy SELECT(WR(j),WI(j)) = .TRUE. after ordering, since ordering may change the value of complex eigenvalues (especially if the eigenvalue is ill-conditioned); in this case INFO may be set to N+3 (see INFO below).

SENSE Input parameter.

SENSE is CHARACTER*1

Determines which reciprocal condition numbers are computed. = 'N': None are computed; = 'E': Computed for average of selected eigenvalues only; = 'V': Computed for selected right invariant subspace only; = 'B': Computed for both. If SENSE = 'E', 'V' or 'B', SORT must equal 'S'.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A. On exit, A is overwritten by its real Schur form T.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

SDIM Output parameter.

SDIM is INTEGER

If SORT = 'N', SDIM = 0. If SORT = 'S', SDIM = number of eigenvalues (after sorting) for which SELECT is true. (Complex conjugate pairs for which SELECT is true for either eigenvalue count as 2.)

WR Output parameter.

WR is DOUBLE PRECISION

WR is an array, dimension (N) .

WI Output parameter.

WI is DOUBLE PRECISION

WI is an array, dimension (N). WR and WI contain the real and imaginary parts, respectively, of the computed eigenvalues, in the same order that they appear on the diagonal of the output Schur form T. Complex conjugate pairs of eigenvalues appear consecutively with the eigenvalue having the positive imaginary part first.

VS Output parameter.

VS is DOUBLE PRECISION

VS is an array, dimension (LDVS, N). If JOBVS = 'V', VS contains the orthogonal matrix Z of Schur vectors. If JOBVS = 'N', VS is not referenced.

LDVS Input parameter.

LDVS is INTEGER

The leading dimension of the array VS. LDVS >= 1, and if JOBVS = 'V', LDVS >= N.

RCONDE Output parameter.

RCONDE is DOUBLE PRECISION

If SENSE = 'E' or 'B', RCONDE contains the reciprocal condition number for the average of the selected eigenvalues. Not referenced if SENSE = 'N' or 'V'.

RCONDV Output parameter.

RCONDV is DOUBLE PRECISION

If SENSE = 'V' or 'B', RCONDV contains the reciprocal condition number for the selected right invariant subspace. Not referenced if SENSE = 'N' or 'E'.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,3*N). Also, if SENSE = 'E' or 'V' or 'B', LWORK >= N+2*SDIM*(N-SDIM), where SDIM is the number of selected eigenvalues computed by this routine. Note

that N+2*SDIM*(N-SDIM) <= N+N*N/2. Note also that an error is only returned if LWORK < max(1,3*N), but if SENSE = 'E' or 'V' or 'B' this may not be large enough. For good performance, LWORK must generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates upper bounds on the optimal sizes of the arrays WORK and IWORK, returns these values as the first entries of the WORK and IWORK arrays, and no error messages related to LWORK or LIWORK are issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. LIWORK >= 1; if SENSE = 'V' or 'B', LIWORK >= SDIM*(N-SDIM). Note that SDIM*(N-SDIM) <= N*N/4. Note also that an error is only returned if LIWORK < 1, but if SENSE = 'V' or 'B' this may not be large enough.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates upper bounds on the optimal sizes of the arrays WORK and IWORK, returns these values as the first entries of the WORK and IWORK arrays, and no error messages related to LWORK or LIWORK are issued by XERBLA.

BWORK Output parameter.

BWORK is LOGICAL

BWORK is an array, dimension (N). Not referenced if SORT = 'N'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, and i is <= N: the QR algorithm failed to compute all the eigenvalues; elements 1:ILO-1 and i+1:N of WR and WI contain those eigenvalues which have converged; if JOBVS = 'V', VS contains the transformation which reduces A to its partially converged Schur form. = N+1: the eigenvalues could not be reordered because some eigenvalues were too close to separate (the problem is very ill-conditioned); = N+2: after reordering, roundoff changed values of some complex eigenvalues so that leading eigenvalues in the Schur form no longer satisfy SELECT=.TRUE. This could also be caused by underflow due to scaling.

Related Information

For this routine in other precisions, please see *cgeesx*, *sgeesx* and *zgeesx*. It also exists with a native C interface as *LAPACKE_dgeesx*.

4.13.21 dgeev

dgeev computes for an N-by-N real nonsymmetric matrix A, the eigenvalues and, optionally, the left and/or right eigenvectors.

The right eigenvector v(j) of A satisfies

```
A * v(j) = lambda(j) * v(j)
```

where lambda(j) is its eigenvalue. The left eigenvector u(j) of A satisfies

```
u(j)**H * A = lambda(j) * u(j)**H
```

where $u(j)^H$ denotes the conjugate-transpose of u(j).

The computed eigenvectors are normalized to have Euclidean norm equal to 1 and largest component real.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgeev(JOBVL, JOBVR, N, A, LDA, WR, WI, VL, LDVL, VR, LDVR, WORK,
                 LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgeev_(const char *jobvl, const char *jobvr, const armpl_int_t *n,
            double *a, const armpl_int_t *lda, double *wr, double *wi,
            double *vl, const armpl_int_t *ldvl, double *vr,
            const armpl_int_t *ldvr, double *work, const armpl_int_t *lwork,
            armpl_int_t *info, ... );
```

Parameters

JOBVL Input parameter.

JOBVL is CHARACTER*1

= 'N': left eigenvectors of A are not computed; = 'V': left eigenvectors of A are computed.

JOBVR Input parameter.

JOBVR is CHARACTER*1

= 'N': right eigenvectors of A are not computed; = 'V': right eigenvectors of A are computed.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A. On exit, A has been overwritten.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

WR Output parameter.

WR is DOUBLE PRECISION

WR is an array, dimension (N) .

WI Output parameter.

WI is DOUBLE PRECISION

WI is an array, dimension (N). WR and WI contain the real and imaginary parts, respectively, of the computed eigenvalues. Complex conjugate pairs of eigenvalues appear consecutively with the eigenvalue having the positive imaginary part first.

VL Output parameter.

VL is DOUBLE PRECISION

VL is an array, dimension (LDVL, N). If JOBVL = 'V', the left eigenvectors u(j) are stored one after another in the columns of VL, in the same order as their eigenvalues. If JOBVL = 'N', VL is not referenced. If the j-th eigenvalue is real, then u(j) = VL(:,j), the j-th column of VL. If the j-th and (j+1)-st eigenvalues form a complex conjugate pair, then u(j) = VL(:,j) + i*VL(:,j+1) and u(j+1) = VL(:,j) - i*VL(:,j+1).

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the array VL. LDVL >= 1; if JOBVL = 'V', LDVL >= N.

VR Output parameter.

VR is DOUBLE PRECISION

VR is an array, dimension (LDVR, N). If JOBVR = 'V', the right eigenvectors v(j) are stored one after another in the columns of VR, in the same order as their eigenvalues. If JOBVR = 'N', VR is not referenced. If the j-th eigenvalue is real, then v(j) = VR(:,j), the j-th column of VR. If the j-th and (j+1)-st eigenvalues form a complex conjugate pair, then v(j) = VR(:,j) + i*VR(:,j+1) and v(j+1) = VR(:,j) - i*VR(:,j+1).

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the array VR. LDVR >= 1; if JOBVR = 'V', LDVR >= N.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,3*N), and if JOBVL = 'V' or JOBVR = 'V', LWORK >= 4*N. For good performance, LWORK must generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, the QR algorithm failed to compute all the eigenvalues, and no eigenvectors have been computed; elements i+1:N of WR and WI contain eigenvalues which have converged.

Related Information

For this routine in other precisions, please see *cgeev*, *sgeev* and *zgeev*. It also exists with a native C interface as *LAPACKE_dgeev*.

4.13.22 dgeevx

dgeevx computes for an N-by-N real nonsymmetric matrix A, the eigenvalues and, optionally, the left and/or right eigenvectors.

Optionally also, it computes a balancing transformation to improve the conditioning of the eigenvalues and eigenvectors (ILO, IHI, SCALE, and ABNRM), reciprocal condition numbers for the eigenvalues (RCONDE), and reciprocal condition numbers for the right eigenvectors (RCONDV).

The right eigenvector v(j) of A satisfies

```
A * v(j) = lambda(j) * v(j)
```

where lambda(j) is its eigenvalue. The left eigenvector u(j) of A satisfies

```
u(j)**H * A = lambda(j) * u(j)**H
```

where $u(j)^H$ denotes the conjugate-transpose of u(j).

The computed eigenvectors are normalized to have Euclidean norm equal to 1 and largest component real.

Balancing a matrix means permuting the rows and columns to make it more nearly upper triangular, and applying a diagonal similarity transformation D * A * D**(-1), where D is a diagonal matrix, to make its rows and columns closer in norm and the condition numbers of its eigenvalues and eigenvectors smaller. The computed reciprocal condition numbers correspond to the balanced matrix. Permuting rows and columns will not change the condition numbers (in exact arithmetic) but diagonal scaling will. For further explanation of balancing, see section 4.10.2 of the LAPACK Users' Guide.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgeevx(BALANC, JOBVL, JOBVR, SENSE, N, A, LDA, WR, WI, VL, LDVL,
                  VR, LDVR, ILO, IHI, SCALE, ABNRM, RCONDE, RCONDV, WORK,
                  LWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgeevx_(const char *balanc, const char *jobvl, const char *jobvr,
             const char *sense, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, double *wr, double *wi, double *vl,
             const armpl_int_t *ldvl, double *vr, const armpl_int_t *ldvr,
             armpl_int_t *ilo, armpl_int_t *ihi, double *scale, double *abnrm,
             double *rconde, double *rcondv, double *work,
             const armpl_int_t *lwork, armpl_int_t *iwork, armpl_int_t *info,
             ... );
```

Parameters

BALANC Input parameter.

BALANC is CHARACTER*1

Indicates how the input matrix should be diagonally scaled and/or permuted to improve the conditioning of its eigenvalues. = 'N': Do not diagonally scale or permute; = 'P': Perform permutations to make the matrix more nearly upper triangular. Do not diagonally scale; = 'S': Diagonally scale the matrix, i.e. replace A by D*A*D**(-1), where D is a diagonal matrix chosen to make the rows and columns of A more equal in norm. Do not permute; = 'B': Both diagonally scale and permute A.

Computed reciprocal condition numbers will be for the matrix after balancing and/or permuting. Permuting does not change condition numbers (in exact arithmetic), but balancing does.

JOBVL Input parameter.

JOBVL is CHARACTER*1

= 'N': left eigenvectors of A are not computed; = 'V': left eigenvectors of A are computed. If SENSE = 'E' or 'B', JOBVL must = 'V'.

JOBVR Input parameter.

JOBVR is CHARACTER*1

= 'N': right eigenvectors of A are not computed; = 'V': right eigenvectors of A are computed. If SENSE = 'E' or 'B', JOBVR must = 'V'.

SENSE Input parameter.

SENSE is CHARACTER*1

Determines which reciprocal condition numbers are computed. = 'N': None are computed; = 'E': Computed for eigenvalues only; = 'V': Computed for right eigenvectors only; = 'B': Computed for eigenvalues and right eigenvectors.

If SENSE = 'E' or 'B', both left and right eigenvectors must also be computed (JOBVL = 'V' and JOBVR = 'V').

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A. On exit, A has been overwritten. If JOBVL = 'V' or JOBVR = 'V', A contains the real Schur form of the balanced version of the input matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

WR Output parameter.

WR is DOUBLE PRECISION

WR is an array, dimension (N) .

WI Output parameter.

WI is DOUBLE PRECISION

WI is an array, dimension (N). WR and WI contain the real and imaginary parts, respectively, of the computed eigenvalues. Complex conjugate pairs of eigenvalues will appear consecutively with the eigenvalue having the positive imaginary part first.

VL Output parameter.

VL is DOUBLE PRECISION

VL is an array, dimension (LDVL, N). If JOBVL = 'V', the left eigenvectors u(j) are stored one after another in the columns of VL, in the same order as their eigenvalues. If JOBVL = 'N', VL is not referenced. If the j-th eigenvalue is real, then u(j) = VL(:,j), the j-th column of VL. If the j-th and (j+1)-st eigenvalues form a complex conjugate pair, then u(j) = VL(:,j) + i*VL(:,j+1) and u(j+1) = VL(:,j) - i*VL(:,j+1).

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the array VL. LDVL >= 1; if JOBVL = 'V', LDVL >= N.

VR Output parameter.

VR is DOUBLE PRECISION

VR is an array, dimension (LDVR, N). If JOBVR = 'V', the right eigenvectors v(j) are stored one after another in the columns of VR, in the same order as their eigenvalues. If JOBVR = 'N', VR is not referenced. If the

j-th eigenvalue is real, then v(j) = VR(:,j), the j-th column of VR. If the j-th and (j+1)-st eigenvalues form a complex conjugate pair, then v(j) = VR(:,j) + i*VR(:,j+1) and v(j+1) = VR(:,j) - i*VR(:,j+1).

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the array VR. LDVR >= 1, and if JOBVR = 'V', LDVR >= N.

ILO Output parameter.

ILO is INTEGER

IHI Output parameter.

IHI is INTEGER

ILO and IHI are integer values determined when A was balanced. The balanced A(i,j) = 0 if I > J and J = 1,...,ILO-1 or I = IHI+1,...,N.

SCALE Output parameter.

SCALE is DOUBLE PRECISION

SCALE is an array, dimension (N). Details of the permutations and scaling factors applied when balancing A. If P(j) is the index of the row and column interchanged with row and column j, and D(j) is the scaling factor applied to row and column j, then SCALE(J) = P(J), for J = 1,...,ILO-1 = D(J), for J = ILO,...,IHI = P(J) for J = IHI+1,...,N. The order in which the interchanges are made is N to IHI+1, then 1 to ILO-1.

ABNRM Output parameter.

ABNRM is DOUBLE PRECISION

The one-norm of the balanced matrix (the maximum of the sum of absolute values of elements of any column).

RCONDE Output parameter.

RCONDE is DOUBLE PRECISION

RCONDE is an array, dimension (N). RCONDE(j) is the reciprocal condition number of the j-th eigenvalue.

RCONDV Output parameter.

RCONDV is DOUBLE PRECISION

RCONDV is an array, dimension (N). RCONDV(j) is the reciprocal condition number of the j-th right eigenvector.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SENSE = 'N' or 'E', LWORK >= max(1,2*N), and if JOBVL = 'V' or JOBVR = 'V', LWORK >= 3*N. If SENSE = 'V' or 'B', LWORK >= N*(N+6). For good performance, LWORK must generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (2*N-2)

If SENSE = 'N' or 'E', not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, the QR algorithm failed to compute all the eigenvalues, and no eigenvectors or condition numbers have been computed; elements 1:ILO-1 and i+1:N of WR and WI contain eigenvalues which have converged.

Related Information

For this routine in other precisions, please see *cgeevx*, *sgeevx* and *zgeevx*. It also exists with a native C interface as *LAPACKE_dgeevx*.

4.13.23 dgehrd

dgehrd reduces a real general matrix A to upper Hessenberg form H by an orthogonal similarity transformation: $Q^T * A * Q = H$.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgehrd(N, ILO, IHI, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgehrd_(const armpl_int_t *n, const armpl_int_t *ilo,
             const armpl_int_t *ihi, double *a, const armpl_int_t *lda,
             double *tau, double *work, const armpl_int_t *lwork,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

It is assumed that A is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N. ILO and IHI are normally set by a previous call to DGEBAL; otherwise they should be set to 1 and N respectively. See Further Details. 1 <= ILO <= IHI <= N, if N > 0; ILO=1 and IHI=0, if N=0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the N-by-N general matrix to be reduced. On exit, the upper triangle and the first subdiagonal of A are overwritten with the upper Hessenberg matrix H, and the elements

below the first subdiagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TAU Output parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (N-1). The scalar factors of the elementary reflectors (see Further Details). Elements 1:ILO-1 and IHI:N-1 of TAU are set to zero.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (LWORK). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= max(1, N). For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cgehrd*, *sgehrd* and *zgehrd*. It also exists with a native C interface as *LAPACKE_dgehrd*.

4.13.24 dhsein

dhsein uses inverse iteration to find specified right and/or left eigenvectors of a real upper Hessenberg matrix H.

The right eigenvector x and the left eigenvector y of the matrix H corresponding to an eigenvalue w are defined by:

```
H * x = w * x,     y**h * H = w * y**h
```

where y**h denotes the conjugate transpose of the vector y.

Syntax

Fortran specification:

```
use armpl_library

subroutine dhsein(SIDE, EIGSRC, INITV, SELECT, N, H, LDH, WR, WI, VL, LDVL,
                  VR, LDVR, MM, M, WORK, IFAILL, IFAILR, INFO)
```

C specification:

```
#include "armpl.h"

void dhsein_(const char *side, const char *eigsrc, const char *initv,
             armpl_int_t *select, const armpl_int_t *n, const double *h,
             const armpl_int_t *ldh, double *wr, const double *wi, double *vl,
             const armpl_int_t *ldvl, double *vr, const armpl_int_t *ldvr,
             const armpl_int_t *mm, armpl_int_t *m, double *work,
             armpl_int_t *ifaill, armpl_int_t *ifailr, armpl_int_t *info,
             ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'R': compute right eigenvectors only; = 'L': compute left eigenvectors only; = 'B': compute both right and left eigenvectors.

EIGSRC Input parameter.

EIGSRC is CHARACTER*1

Specifies the source of eigenvalues supplied in (WR, WI): = 'Q': the eigenvalues were found using DHSEQR; thus, if H has zero subdiagonal elements, and so is block-triangular, then the j-th eigenvalue can be assumed to be an eigenvalue of the block containing the j-th row/column. This property allows DHSEIN to perform inverse iteration on just one diagonal block. = 'N': no assumptions are made on the correspondence between eigenvalues and diagonal blocks. In this case, DHSEIN must always perform inverse iteration using the whole matrix H.

INITV Input parameter.

INITV is CHARACTER*1

= 'N': no initial vectors are supplied; = 'U': user-supplied initial vectors are stored in the arrays VL and/or VR.

SELECT Input and output parameter.

SELECT is LOGICAL

SELECT is an array, dimension (N). Specifies the eigenvectors to be computed. To select the real eigenvector corresponding to a real eigenvalue WR(j), SELECT(j) must be set to .TRUE.. To select the complex eigenvector corresponding to a complex eigenvalue (WR(j),WI(j)), with complex conjugate (WR(j+1),WI(j+1)), either SELECT(j) or SELECT(j+1) or both must be set to .TRUE.; then on exit SELECT(j) is .TRUE. and SELECT(j+1) is .FALSE..

N Input parameter.

N is INTEGER

The order of the matrix H. N >= 0.

H Input parameter.

H is DOUBLE PRECISION

H is an array, dimension (LDH, N). The upper Hessenberg matrix H. If a NaN is detected in H, the routine will return with INFO=-6.

LDH Input parameter.

LDH is INTEGER

The leading dimension of the array H. LDH >= max(1, N).

WR Input and output parameter.

WR is DOUBLE PRECISION

WR is an array, dimension (N) .

WI Input parameter.

WI is DOUBLE PRECISION

WI is an array, dimension (N). On entry, the real and imaginary parts of the eigenvalues of H; a complex conjugate pair of eigenvalues must be stored in consecutive elements of WR and WI. On exit, WR may have been altered since close eigenvalues are perturbed slightly in searching for independent eigenvectors.

VL Input and output parameter.

VL is DOUBLE PRECISION

VL is an array, dimension (LDVL, MM). On entry, if INITV = 'U' and SIDE = 'L' or 'B', VL must contain starting vectors for the inverse iteration for the left eigenvectors; the starting vector for each eigenvector must be in the same column(s) in which the eigenvector will be stored. On exit, if SIDE = 'L' or 'B', the left eigenvectors specified by SELECT will be stored consecutively in the columns of VL, in the same order as their eigenvalues. A complex eigenvector corresponding to a complex eigenvalue is stored in two consecutive columns, the first holding the real part and the second the imaginary part. If SIDE = 'R', VL is not referenced.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the array VL. LDVL >= max(1, N) if SIDE = 'L' or 'B'; LDVL >= 1 otherwise.

VR Input and output parameter.

VR is DOUBLE PRECISION

VR is an array, dimension (LDVR, MM). On entry, if INITV = 'U' and SIDE = 'R' or 'B', VR must contain starting vectors for the inverse iteration for the right eigenvectors; the starting vector for each eigenvector must be in the same column(s) in which the eigenvector will be stored. On exit, if SIDE = 'R' or 'B', the right eigenvectors specified by SELECT will be stored consecutively in the columns of VR, in the same order as their eigenvalues. A complex eigenvector corresponding to a complex eigenvalue is stored in two consecutive columns, the first holding the real part and the second the imaginary part. If SIDE = 'L', VR is not referenced.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the array VR. LDVR >= max(1, N) if SIDE = 'R' or 'B'; LDVR >= 1 otherwise.

MM Input parameter.

MM is INTEGER

The number of columns in the arrays VL and/or VR. MM >= M.

M Output parameter.

M is INTEGER

The number of columns in the arrays VL and/or VR required to store the eigenvectors; each selected real eigenvector occupies one column and each selected complex eigenvector occupies two columns.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension ((N+2)*N) .

IFAILL Output parameter.

IFAILL is INTEGER array, dimension (MM)

If SIDE = 'L' or 'B', IFAILL(i) = j > 0 if the left eigenvector in the i-th column of VL (corresponding to the eigenvalue w(j)) failed to converge; IFAILL(i) = 0 if the eigenvector converged satisfactorily. If the i-th and (i+1)th columns of VL hold a complex eigenvector, then IFAILL(i) and IFAILL(i+1) are set to the same value. If SIDE = 'R', IFAILL is not referenced.

IFAILR Output parameter.

IFAILR is INTEGER array, dimension (MM)

If SIDE = 'R' or 'B', IFAILR(i) = j > 0 if the right eigenvector in the i-th column of VR (corresponding to the eigenvalue w(j)) failed to converge; IFAILR(i) = 0 if the eigenvector converged satisfactorily. If the i-th and (i+1)th columns of VR hold a complex eigenvector, then IFAILR(i) and IFAILR(i+1) are set to the same value. If SIDE = 'L', IFAILR is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, i is the number of eigenvectors which failed to converge; see IFAILL and IFAILR for further details.

Related Information

For this routine in other precisions, please see *chsein*, *shsein* and *zhsein*. It also exists with a native C interface as *LAPACKE_dhsein*.

4.13.25 dhseqr

```
DHSEQR computes the eigenvalues of a Hessenberg matrix H
   and, optionally, the matrices T and Z from the Schur decomposition
   H = Z T Z**T, where T is an upper quasi-triangular matrix (the
   Schur form), and Z is the orthogonal matrix of Schur vectors.

   Optionally Z may be postmultiplied into an input orthogonal
   matrix Q so that this routine can give the Schur factorization
   of a matrix A which has been reduced to the Hessenberg form H
   by the orthogonal matrix Q:  A = Q*H*Q**T = (QZ)*T*(QZ)**T.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dhseqr(JOB, COMPZ, N, ILO, IHI, H, LDH, WR, WI, Z, LDZ, WORK,
                  LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dhseqr_(const char *job, const char *compz, const armpl_int_t *n,
             const armpl_int_t *ilo, const armpl_int_t *ihi, double *h,
             const armpl_int_t *ldh, double *wr, double *wi, double *z,
             const armpl_int_t *ldz, double *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

JOB Input parameter.

JOB is CHARACTER*1

= 'E': compute eigenvalues only; = 'S': compute eigenvalues and the Schur form T.

COMPZ Input parameter.

COMPZ is CHARACTER*1

= 'N': no Schur vectors are computed; = 'I': Z is initialized to the unit matrix and the matrix Z of Schur vectors of H is returned; = 'V': Z must contain an orthogonal matrix Q on entry, and the product Q* Z is returned.

N Input parameter.

N is INTEGER

The order of the matrix H. N .GE. 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

It is assumed that H is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N. ILO and IHI are normally set by a previous call to DGEBAL, and then passed to ZGEHRD when the matrix output by DGEBAL is reduced to Hessenberg form. Otherwise ILO and IHI should be set to 1 and N respectively. If N.GT.0, then 1.LE.ILO.LE.IHI.LE.N. If N = 0, then ILO = 1 and IHI = 0.

H Input and output parameter.

H is DOUBLE PRECISION

H is an array, dimension (LDH, N). On entry, the upper Hessenberg matrix H. On exit, if INFO = 0 and JOB = 'S', then H contains the upper quasi-triangular matrix T from the Schur decomposition (the Schur form); 2-by-2 diagonal blocks (corresponding to complex conjugate pairs of eigenvalues) are returned in standard form, with H(i,i) = H(i+1,i+1) and H(i+1,i)*H(i,i+1).LT.0. If INFO = 0 and JOB = 'E', the contents of H are unspecified on exit. (The output value of H when INFO.GT.0 is given under the description of INFO below.)

Unlike earlier versions of DHSEQR, this subroutine may explicitly H(i,j) = 0 for i.GT.j and j = 1, 2, ... ILO-1 or j = IHI+1, IHI+2, ... N.

LDH Input parameter.

LDH is INTEGER

The leading dimension of the array H. LDH .GE. max(1, N).

WR Output parameter.

WR is DOUBLE PRECISION

WR is an array, dimension (N) .

WI Output parameter.

WI is DOUBLE PRECISION

WI is an array, dimension (N). The real and imaginary parts, respectively, of the computed eigenvalues. If two eigenvalues are computed as a complex conjugate pair, they are stored in consecutive elements of WR and WI, say the i-th and (i+1)th, with WI(i) .GT. 0 and WI(i+1) .LT. 0. If JOB = 'S', the eigenvalues are stored in the same order as on the diagonal of the Schur form returned in H, with WR(i) = H(i,i) and, if H(i:i+1,i:i+1) is a 2-by-2 diagonal block, WI(i) = sqrt(-H(i+1,i)*H(i,i+1)) and WI(i+1) = -WI(i).

Z Input and output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, N). If COMPZ = 'N', Z is not referenced. If COMPZ = 'I', on entry Z need not be set and on exit, if INFO = 0, Z contains the orthogonal matrix Z of the Schur vectors of H. If COMPZ = 'V', on entry Z must contain an N-by-N matrix Q, which is assumed to be equal to the unit matrix except for the submatrix Z(ILO:IHI,ILO:IHI). On exit, if INFO = 0, Z contains Q*Z. Normally Q is the orthogonal matrix generated by DORGHR after the call to DGEHRD which formed the Hessenberg matrix H. (The output value of Z when INFO.GT.0 is given under the description of INFO below.)

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. if COMPZ = 'I' or COMPZ = 'V', then LDZ.GE.MAX(1, N). Otherwize, LDZ.GE.1.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (LWORK). On exit, if INFO = 0, WORK(1) returns an estimate of the optimal value for LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK .GE. max(1, N) is sufficient and delivers very good and sometimes optimal performance. However, LWORK as large as 11*N may be required for optimal performance. A workspace query is recommended to determine the optimal workspace size.

If LWORK = -1, then DHSEQR does a workspace query. In this case, DHSEQR checks the input parameters and estimates the optimal workspace size for the given values of N, ILO and IHI. The estimate is returned in WORK(1). No error message related to LWORK is issued by XERBLA. Neither H nor Z are accessed.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit .LT. 0: if INFO = -i, the i-th argument had an illegal value .GT. 0: if INFO = i, DHSEQR failed to compute all of the eigenvalues. Elements 1:ilo-1 and i+1:n of WR and WI contain those eigenvalues which have been successfully computed. (Failures are rare.)

If INFO .GT. 0 and JOB = 'E', then on exit, the remaining unconverged eigenvalues are the eigen- values of the upper Hessenberg matrix rows and columns ILO through INFO of the final, output value of H.

If INFO .GT. 0 and JOB = 'S', then on exit

(*) (initial value of H)*U = U*(final value of H)

where U is an orthogonal matrix. The final value of H is upper Hessenberg and quasi-triangular in rows and columns INFO+1 through IHI.

If INFO .GT. 0 and COMPZ = 'V', then on exit

(final value of Z) = (initial value of Z)*U

where U is the orthogonal matrix in (*) (regard- less of the value of JOB.)

If INFO .GT. 0 and COMPZ = 'I', then on exit (final value of Z) = U where U is the orthogonal matrix in (*) (regard- less of the value of JOB.)

If INFO .GT. 0 and COMPZ = 'N', then Z is not accessed.

Related Information

For this routine in other precisions, please see *chseqr*, *shseqr* and *zhseqr*. It also exists with a native C interface as *LAPACKE_dhseqr*.

4.13.26 dorghr

`dorghr` generates a real orthogonal matrix Q which is defined as the product of IHI-ILO elementary reflectors of order N, as returned by DGEHRD:

Q = H(ilo) H(ilo+1) . . . H(ihi-1).

Syntax

Fortran specification:

```
use armpl_library

subroutine dorghr(N, ILO, IHI, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dorghr_(const armpl_int_t *n, const armpl_int_t *ilo,
             const armpl_int_t *ihi, double *a, const armpl_int_t *lda,
             const double *tau, double *work, const armpl_int_t *lwork,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix Q. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

ILO and IHI must have the same values as in the previous call of DGEHRD. Q is equal to the unit matrix except in the submatrix Q(ilo+1:ihi,ilo+1:ihi). 1 <= ILO <= IHI <= N, if N > 0; ILO=1 and IHI=0, if N=0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the vectors which define the elementary reflectors, as returned by DGEHRD. On exit, the N-by-N orthogonal matrix Q.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TAU Input parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (N-1). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by DGEHRD.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= IHI-ILO. For optimum performance LWORK >= (IHI-ILO)*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *sorghr*. It also exists with a native C interface as *LAPACKE_dorghr*.

4.13.27 dormhr

dormhr overwrites the general real M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'T': Q^T * C C * Q^T

where Q is a real orthogonal matrix of order nq, with nq = m if SIDE = 'L' and nq = n if SIDE = 'R'. Q is defined as the product of IHI-ILO elementary reflectors, as returned by DGEHRD:

Q = H(ilo) H(ilo+1) . . . H(ihi-1).

Syntax

Fortran specification:

```
use armpl_library

subroutine dormhr(SIDE, TRANS, M, N, ILO, IHI, A, LDA, TAU, C, LDC, WORK,
                  LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dormhr_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *ilo,
```

(continues on next page)

(continued from previous page)

```
                const armpl_int_t *ihi, const double *a, const armpl_int_t *lda,
                const double *tau, double *c, const armpl_int_t *ldc,
                double *work, const armpl_int_t *lwork, armpl_int_t *info,
                ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'T': Transpose, apply Q^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

ILO and IHI must have the same values as in the previous call of DGEHRD. Q is equal to the unit matrix except in the submatrix Q(ilo+1:ihi,ilo+1:ihi). If SIDE = 'L', then 1 <= ILO <= IHI <= M, if M > 0, and ILO = 1 and IHI = 0, if M = 0; if SIDE = 'R', then 1 <= ILO <= IHI <= N, if N > 0, and ILO = 1 and IHI = 0, if N = 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension. (LDA, M) if SIDE = 'L' (LDA, N) if SIDE = 'R' The vectors which define the elementary reflectors, as returned by DGEHRD.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M) if SIDE = 'L'; LDA >= max(1, N) if SIDE = 'R'.

TAU Input parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension. (M-1) if SIDE = 'L' (N-1) if SIDE = 'R' TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by DGEHRD.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, N); if SIDE = 'R', LWORK >= max(1, M). For optimum performance LWORK >= N*NB if SIDE = 'L', and LWORK >= M*NB if SIDE = 'R', where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *sormhr*. It also exists with a native C interface as *LAPACKE_dormhr*.

4.13.28 dtrevc

dtrevc computes some or all of the right and/or left eigenvectors of a real upper quasi-triangular matrix T. Matrices of this type are produced by the Schur factorization of a real general matrix: A = Q*T*Q^T , as computed by DHSEQR.

The right eigenvector x and the left eigenvector y of T corresponding to an eigenvalue w are defined by:

```
T*x = w*x,     (y**H)*T = w*(y**H)
```

where y^H denotes the conjugate transpose of y. The eigenvalues are not input to this routine, but are read directly from the diagonal blocks of T.

This routine returns the matrices X and/or Y of right and left eigenvectors of T, or the products Q*X and/or Q*Y, where Q is an input matrix. If Q is the orthogonal factor that reduces a matrix A to Schur form T, then Q*X and Q*Y are the matrices of right and left eigenvectors of A.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtrevc(SIDE, HOWMNY, SELECT, N, T, LDT, VL, LDVL, VR, LDVR, MM, M,
                  WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dtrevc_(const char *side, const char *howmny, armpl_int_t *select,
             const armpl_int_t *n, const double *t, const armpl_int_t *ldt,
             double *vl, const armpl_int_t *ldvl, double *vr,
             const armpl_int_t *ldvr, const armpl_int_t *mm, armpl_int_t *m,
             double *work, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'R': compute right eigenvectors only; = 'L': compute left eigenvectors only; = 'B': compute both right and left eigenvectors.

HOWMNY Input parameter.

HOWMNY is CHARACTER*1

= 'A': compute all right and/or left eigenvectors; = 'B': compute all right and/or left eigenvectors, backtransformed by the matrices in VR and/or VL; = 'S': compute selected right and/or left eigenvectors, as indicated by the logical array SELECT.

SELECT Input and output parameter.

SELECT is LOGICAL

SELECT is an array, dimension (N). If HOWMNY = 'S', SELECT specifies the eigenvectors to be computed. If w(j) is a real eigenvalue, the corresponding real eigenvector is computed if SELECT(j) is .TRUE.. If w(j) and w(j+1) are the real and imaginary parts of a complex eigenvalue, the corresponding complex eigenvector is computed if either SELECT(j) or SELECT(j+1) is .TRUE., and on exit SELECT(j) is set to .TRUE. and SELECT(j+1) is set to .FALSE.. Not referenced if HOWMNY = 'A' or 'B'.

N Input parameter.

N is INTEGER

The order of the matrix T. N >= 0.

T Input parameter.

T is DOUBLE PRECISION

T is an array, dimension (LDT, N). The upper quasi-triangular matrix T in Schur canonical form.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

VL Input and output parameter.

VL is DOUBLE PRECISION

VL is an array, dimension (LDVL, MM). On entry, if SIDE = 'L' or 'B' and HOWMNY = 'B', VL must contain an N-by-N matrix Q (usually the orthogonal matrix Q of Schur vectors returned by DHSEQR). On exit, if SIDE = 'L' or 'B', VL contains: if HOWMNY = 'A', the matrix Y of left eigenvectors of T; if HOWMNY = 'B', the matrix Q*Y; if HOWMNY = 'S', the left eigenvectors of T specified by SELECT, stored consecutively in the columns of VL, in the same order as their eigenvalues. A complex eigenvector corresponding to a complex eigenvalue is stored in two consecutive columns, the first holding the real part, and the second the imaginary part. Not referenced if SIDE = 'R'.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the array VL. LDVL >= 1, and if SIDE = 'L' or 'B', LDVL >= N.

VR Input and output parameter.

VR is DOUBLE PRECISION

VR is an array, dimension (LDVR, MM). On entry, if SIDE = 'R' or 'B' and HOWMNY = 'B', VR must contain an N-by-N matrix Q (usually the orthogonal matrix Q of Schur vectors returned by DHSEQR). On exit, if SIDE = 'R' or 'B', VR contains: if HOWMNY = 'A', the matrix X of right eigenvectors of T; if HOWMNY = 'B', the matrix Q*X; if HOWMNY = 'S', the right eigenvectors of T specified by SELECT, stored consecutively in the columns of VR, in the same order as their eigenvalues. A complex eigenvector corresponding to a complex eigenvalue is stored in two consecutive columns, the first holding the real part and the second the imaginary part. Not referenced if SIDE = 'L'.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the array VR. LDVR >= 1, and if SIDE = 'R' or 'B', LDVR >= N.

MM Input parameter.

MM is INTEGER

The number of columns in the arrays VL and/or VR. MM >= M.

M Output parameter.

M is INTEGER

The number of columns in the arrays VL and/or VR actually used to store the eigenvectors. If HOWMNY = 'A' or 'B', M is set to N. Each selected real eigenvector occupies one column and each selected complex eigenvector occupies two columns.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (3*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctrevc*, *strevc* and *ztrevc*. It also exists with a native C interface as *LAPACKE_dtrevc*.

4.13.29 dtrexc

dtrexc reorders the real Schur factorization of a real matrix A = Q*T*Q^T , so that the diagonal block of T with row index IFST is moved to row ILST.

The real Schur form T is reordered by an orthogonal similarity transformation Z^T *T*Z, and optionally the matrix Q of Schur vectors is updated by postmultiplying it with Z.

T must be in Schur canonical form (as returned by DHSEQR), that is, block upper triangular with 1-by-1 and 2-by-2 diagonal blocks; each 2-by-2 diagonal block has its diagonal elements equal and its off-diagonal elements of opposite sign.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtrexc(COMPQ, N, T, LDT, Q, LDQ, IFST, ILST, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dtrexc_(const char *compq, const armpl_int_t *n, double *t,
             const armpl_int_t *ldt, double *q, const armpl_int_t *ldq,
             armpl_int_t *ifst, armpl_int_t *ilst, double *work,
             armpl_int_t *info, ... );
```

Parameters

COMPQ Input parameter.

COMPQ is CHARACTER*1

= 'V': update the matrix Q of Schur vectors; = 'N': do not update Q.

N Input parameter.

N is INTEGER

The order of the matrix T. N >= 0. If N == 0 arguments ILST and IFST may be any value.

T Input and output parameter.

T is DOUBLE PRECISION

T is an array, dimension (LDT, N). On entry, the upper quasi-triangular matrix T, in Schur Schur canonical form. On exit, the reordered upper quasi-triangular matrix, again in Schur canonical form.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

Q Input and output parameter.

Q is DOUBLE PRECISION

Q is an array, dimension (LDQ, N). On entry, if COMPQ = 'V', the matrix Q of Schur vectors. On exit, if COMPQ = 'V', Q has been postmultiplied by the orthogonal transformation matrix Z which reorders T. If COMPQ = 'N', Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= 1, and if COMPQ = 'V', LDQ >= max(1, N).

IFST Input and output parameter.

IFST is INTEGER

ILST Input and output parameter.

ILST is INTEGER

Specify the reordering of the diagonal blocks of T. The block with row index IFST is moved to row ILST, by a sequence of transpositions between adjacent blocks. On exit, if IFST pointed on entry to the second

row of a 2-by-2 block, it is changed to point to the first row; ILST always points to the first row of the block in its final position (which may differ from its input value by +1 or -1). 1 <= IFST <= N; 1 <= ILST <= N.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value = 1: two adjacent blocks were too close to swap (the problem is very ill-conditioned); T may have been partially reordered, and ILST points to the first row of the current position of the block being moved.

Related Information

For this routine in other precisions, please see *ctrexc*, *strexc* and *ztrexc*. It also exists with a native C interface as *LAPACKE_dtrexc*.

4.13.30 dtrsen

dtrsen reorders the real Schur factorization of a real matrix A = Q*T*Q^T , so that a selected cluster of eigenvalues appears in the leading diagonal blocks of the upper quasi-triangular matrix T, and the leading columns of Q form an orthonormal basis of the corresponding right invariant subspace.

Optionally the routine computes the reciprocal condition numbers of the cluster of eigenvalues and/or the invariant subspace.

T must be in Schur canonical form (as returned by DHSEQR), that is, block upper triangular with 1-by-1 and 2-by-2 diagonal blocks; each 2-by-2 diagonal block has its diagonal elements equal and its off-diagonal elements of opposite sign.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtrsen(JOB, COMPQ, SELECT, N, T, LDT, Q, LDQ, WR, WI, M, S, SEP,
                  WORK, LWORK, IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dtrsen_(const char *job, const char *compq, const armpl_int_t *select,
             const armpl_int_t *n, double *t, const armpl_int_t *ldt,
             double *q, const armpl_int_t *ldq, double *wr, double *wi,
             armpl_int_t *m, double *s, double *sep, double *work,
             const armpl_int_t *lwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOB Input parameter.

JOB is CHARACTER*1

Specifies whether condition numbers are required for the cluster of eigenvalues (S) or the invariant subspace (SEP): = 'N': none; = 'E': for eigenvalues only (S); = 'V': for invariant subspace only (SEP); = 'B': for both eigenvalues and invariant subspace (S and SEP).

COMPQ Input parameter.

COMPQ is CHARACTER*1

= 'V': update the matrix Q of Schur vectors; = 'N': do not update Q.

SELECT Input parameter.

SELECT is LOGICAL

SELECT is an array, dimension (N). SELECT specifies the eigenvalues in the selected cluster. To select a real eigenvalue w(j), SELECT(j) must be set to .TRUE.. To select a complex conjugate pair of eigenvalues w(j) and w(j+1), corresponding to a 2-by-2 diagonal block, either SELECT(j) or SELECT(j+1) or both must be set to .TRUE.; a complex conjugate pair of eigenvalues must be either both included in the cluster or both excluded.

N Input parameter.

N is INTEGER

The order of the matrix T. N >= 0.

T Input and output parameter.

T is DOUBLE PRECISION

T is an array, dimension (LDT, N). On entry, the upper quasi-triangular matrix T, in Schur canonical form. On exit, T is overwritten by the reordered matrix T, again in Schur canonical form, with the selected eigenvalues in the leading diagonal blocks.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

Q Input and output parameter.

Q is DOUBLE PRECISION

Q is an array, dimension (LDQ, N). On entry, if COMPQ = 'V', the matrix Q of Schur vectors. On exit, if COMPQ = 'V', Q has been postmultiplied by the orthogonal transformation matrix which reorders T; the leading M columns of Q form an orthonormal basis for the specified invariant subspace. If COMPQ = 'N', Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= 1; and if COMPQ = 'V', LDQ >= N.

WR Output parameter.

WR is DOUBLE PRECISION

WR is an array, dimension (N) .

WI Output parameter.

WI is DOUBLE PRECISION

WI is an array, dimension (N). The real and imaginary parts, respectively, of the reordered eigenvalues of T. The eigenvalues are stored in the same order as on the diagonal of T, with WR(i) = T(i,i) and, if T(i:i+1,i:i+1) is a 2-by-2 diagonal block, WI(i) > 0 and WI(i+1) = -WI(i). Note that if a complex eigenvalue is sufficiently ill-conditioned, then its value may differ significantly from its value before reordering.

M Output parameter.

M is INTEGER

The dimension of the specified invariant subspace. 0 < = M <= N.

S Output parameter.

S is DOUBLE PRECISION

If JOB = 'E' or 'B', S is a lower bound on the reciprocal condition number for the selected cluster of eigenvalues. S cannot underestimate the true reciprocal condition number by more than a factor of sqrt(N). If M = 0 or N, S = 1. If JOB = 'N' or 'V', S is not referenced.

SEP Output parameter.

SEP is DOUBLE PRECISION

If JOB = 'V' or 'B', SEP is the estimated reciprocal condition number of the specified invariant subspace. If M = 0 or N, SEP = norm(T). If JOB = 'N' or 'E', SEP is not referenced.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If JOB = 'N', LWORK >= max(1, N); if JOB = 'E', LWORK >= max(1,M*(N-M)); if JOB = 'V' or 'B', LWORK >= max(1,2*M*(N-M)).

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. If JOB = 'N' or 'E', LIWORK >= 1; if JOB = 'V' or 'B', LIWORK >= max(1,M*(N-M)).

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the IWORK array, returns this value as the first entry of the IWORK array, and no error message related to LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value = 1: reordering of T failed because some eigenvalues are too close to separate (the problem is very ill-conditioned); T may have been partially reordered, and WR and WI contain the eigenvalues in the same order as in T; S and SEP (if requested) are set to zero.

Related Information

For this routine in other precisions, please see *ctrsen*, *strsen* and *ztrsen*. It also exists with a native C interface as *LAPACKE_dtrsen*.

4.13.31 dtrsna

dtrsna estimates reciprocal condition numbers for specified eigenvalues and/or right eigenvectors of a real upper quasi-triangular matrix T (or of any matrix Q*T*Q^T with Q orthogonal).

T must be in Schur canonical form (as returned by DHSEQR), that is, block upper triangular with 1-by-1 and 2-by-2 diagonal blocks; each 2-by-2 diagonal block has its diagonal elements equal and its off-diagonal elements of opposite sign.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtrsna(JOB, HOWMNY, SELECT, N, T, LDT, VL, LDVL, VR, LDVR, S, SEP,
                  MM, M, WORK, LDWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dtrsna_(const char *job, const char *howmny, const armpl_int_t *select,
             const armpl_int_t *n, const double *t, const armpl_int_t *ldt,
             const double *vl, const armpl_int_t *ldvl, const double *vr,
             const armpl_int_t *ldvr, double *s, double *sep,
             const armpl_int_t *mm, armpl_int_t *m, double *work,
             const armpl_int_t *ldwork, armpl_int_t *iwork, armpl_int_t *info,
             ... );
```

Parameters

JOB Input parameter.

JOB is CHARACTER*1

Specifies whether condition numbers are required for eigenvalues (S) or eigenvectors (SEP): = 'E': for eigenvalues only (S); = 'V': for eigenvectors only (SEP); = 'B': for both eigenvalues and eigenvectors (S and SEP).

HOWMNY Input parameter.

HOWMNY is CHARACTER*1

= 'A': compute condition numbers for all eigenpairs; = 'S': compute condition numbers for selected eigenpairs specified by the array SELECT.

SELECT Input parameter.

SELECT is LOGICAL

SELECT is an array, dimension (N). If HOWMNY = 'S', SELECT specifies the eigenpairs for which condition numbers are required. To select condition numbers for the eigenpair corresponding to a real eigenvalue w(j), SELECT(j) must be set to .TRUE.. To select condition numbers corresponding to a complex conjugate pair of eigenvalues w(j) and w(j+1), either SELECT(j) or SELECT(j+1) or both, must be set to .TRUE.. If HOWMNY = 'A', SELECT is not referenced.

N Input parameter.

N is INTEGER

The order of the matrix T. N >= 0.

T Input parameter.

T is DOUBLE PRECISION

T is an array, dimension (LDT, N). The upper quasi-triangular matrix T, in Schur canonical form.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

VL Input parameter.

VL is DOUBLE PRECISION

VL is an array, dimension (LDVL, M). If JOB = 'E' or 'B', VL must contain left eigenvectors of T (or of any Q*T*Q^T with Q orthogonal), corresponding to the eigenpairs specified by HOWMNY and SELECT. The eigenvectors must be stored in consecutive columns of VL, as returned by DHSEIN or DTREVC. If JOB = 'V', VL is not referenced.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the array VL. LDVL >= 1; and if JOB = 'E' or 'B', LDVL >= N.

VR Input parameter.

VR is DOUBLE PRECISION

VR is an array, dimension (LDVR, M). If JOB = 'E' or 'B', VR must contain right eigenvectors of T (or of any Q*T*Q^T with Q orthogonal), corresponding to the eigenpairs specified by HOWMNY and SELECT. The eigenvectors must be stored in consecutive columns of VR, as returned by DHSEIN or DTREVC. If JOB = 'V', VR is not referenced.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the array VR. LDVR >= 1; and if JOB = 'E' or 'B', LDVR >= N.

S Output parameter.

S is DOUBLE PRECISION

S is an array, dimension (MM). If JOB = 'E' or 'B', the reciprocal condition numbers of the selected eigenvalues, stored in consecutive elements of the array. For a complex conjugate pair of eigenvalues two consecutive elements of S are set to the same value. Thus S(j), SEP(j), and the j-th columns of VL and VR all correspond to the same eigenpair (but not in general the j-th eigenpair, unless all eigenpairs are selected). If JOB = 'V', S is not referenced.

SEP Output parameter.

SEP is DOUBLE PRECISION

SEP is an array, dimension (MM). If JOB = 'V' or 'B', the estimated reciprocal condition numbers of the selected eigenvectors, stored in consecutive elements of the array. For a complex eigenvector two consecutive elements of SEP are set to the same value. If the eigenvalues cannot be reordered to compute SEP(j), SEP(j) is set to 0; this can only occur when the true value would be very small anyway. If JOB = 'E', SEP is not referenced.

MM Input parameter.

MM is INTEGER

The number of elements in the arrays S (if JOB = 'E' or 'B') and/or SEP (if JOB = 'V' or 'B'). MM >= M.

M Output parameter.

M is INTEGER

The number of elements of the arrays S and/or SEP actually used to store the estimated condition numbers. If HOWMNY = 'A', M is set to N.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (LDWORK,N+6). If JOB = 'E', WORK is not referenced.

LDWORK Input parameter.

LDWORK is INTEGER

The leading dimension of the array WORK. LDWORK >= 1; and if JOB = 'V' or 'B', LDWORK >= N.

IWORK Output parameter.

IWORK is INTEGER array, dimension (2*(N-1))

If JOB = 'E', IWORK is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctrsna*, *strsna* and *ztrsna*. It also exists with a native C interface as *LAPACKE_dtrsna*.

4.13.32 dtrsyl

dtrsyl solves the real Sylvester matrix equation:

```
op(A)*X + X*op(B) = scale*C or
op(A)*X - X*op(B) = scale*C,
```

where op(A) = A or A^T , and A and B are both upper quasi- triangular. A is M-by-M and B is N-by-N; the right hand side C and the solution X are M-by-N; and scale is an output scale factor, set <= 1 to avoid overflow in X.

A and B must be in Schur canonical form (as returned by DHSEQR), that is, block upper triangular with 1-by-1 and 2-by-2 diagonal blocks; each 2-by-2 diagonal block has its diagonal elements equal and its off-diagonal elements of opposite sign.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtrsyl(TRANA, TRANB, ISGN, M, N, A, LDA, B, LDB, C, LDC, SCALE,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dtrsyl_(const char *trana, const char *tranb, const armpl_int_t *isgn,
             const armpl_int_t *m, const armpl_int_t *n, const double *a,
             const armpl_int_t *lda, const double *b, const armpl_int_t *ldb,
             double *c, const armpl_int_t *ldc, double *scale,
             armpl_int_t *info, ... );
```

Parameters

TRANA Input parameter.

TRANA is CHARACTER*1

Specifies the option op(A): = 'N': op(A) = A (No transpose) = 'T': op(A) = A^T (Transpose) = 'C': op(A) = A^H (Conjugate transpose = Transpose)

TRANB Input parameter.

TRANB is CHARACTER*1

Specifies the option op(B): = 'N': op(B) = B (No transpose) = 'T': op(B) = B^T (Transpose) = 'C': op(B) = B^H (Conjugate transpose = Transpose)

ISGN Input parameter.

ISGN is INTEGER

Specifies the sign in the equation: = +1: solve op(A)*X + X*op(B) = scale*C = -1: solve op(A)*X - X*op(B) = scale* C

M Input parameter.

M is INTEGER

The order of the matrix A, and the number of rows in the matrices X and C. M >= 0.

N Input parameter.

N is INTEGER

The order of the matrix B, and the number of columns in the matrices X and C. N >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, M). The upper quasi-triangular matrix A, in Schur canonical form.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). The upper quasi-triangular matrix B, in Schur canonical form.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, N). On entry, the M-by-N right hand side matrix C. On exit, C is overwritten by the solution matrix X.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M)

SCALE Output parameter.

SCALE is DOUBLE PRECISION

The scale factor, scale, set <= 1 to avoid overflow in X.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value = 1: A and B have common or very close eigenvalues; perturbed values were used to solve the equation (but the matrices A and B are unchanged).

Related Information

For this routine in other precisions, please see *ctrsyl*, *strsyl* and *ztrsyl*. It also exists with a native C interface as *LAPACKE_dtrsyl*.

4.13.33 sgebak

sgebak forms the right or left eigenvectors of a real general matrix by backward transformation on the computed eigenvectors of the balanced matrix output by SGEBAL.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgebak(JOB, SIDE, N, ILO, IHI, SCALE, M, V, LDV, INFO)
```

C specification:

```
#include "armpl.h"

void sgebak_(const char *job, const char *side, const armpl_int_t *n,
             const armpl_int_t *ilo, const armpl_int_t *ihi,
             const float *scale, const armpl_int_t *m, float *v,
             const armpl_int_t *ldv, armpl_int_t *info, ... );
```

Parameters

JOB Input parameter.

JOB is CHARACTER*1

Specifies the type of backward transformation required: = 'N', do nothing, return immediately; = 'P', do backward transformation for permutation only; = 'S', do backward transformation for scaling only; = 'B', do backward transformations for both permutation and scaling. JOB must be the same as the argument JOB supplied to SGEBAL.

SIDE Input parameter.

SIDE is CHARACTER*1

= 'R': V contains right eigenvectors; = 'L': V contains left eigenvectors.

N Input parameter.

N is INTEGER

The number of rows of the matrix V. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

The integers ILO and IHI determined by SGEBAL. 1 <= ILO <= IHI <= N, if N > 0; ILO=1 and IHI=0, if N=0.

SCALE Input parameter.

SCALE is REAL

SCALE is an array, dimension (N). Details of the permutation and scaling factors, as returned by SGEBAL.

M Input parameter.

M is INTEGER

The number of columns of the matrix V. M >= 0.

V Input and output parameter.

V is REAL

V is an array, dimension (LDV, M). On entry, the matrix of right or left eigenvectors to be transformed, as returned by SHSEIN or STREVC. On exit, V is overwritten by the transformed eigenvectors.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. LDV >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cgebak*, *dgebak* and *zgebak*. It also exists with a native C interface as *LAPACKE_sgebak*.

4.13.34 sgebal

sgebal balances a general real matrix A. This involves, first, permuting A by a similarity transformation to isolate eigenvalues in the first 1 to ILO-1 and last IHI+1 to N elements on the diagonal; and second, applying a diagonal similarity transformation to rows and columns ILO to IHI to make the rows and columns as close in norm as possible. Both steps are optional.

Balancing may reduce the 1-norm of the matrix, and improve the accuracy of the computed eigenvalues and/or eigenvectors.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgebal(JOB, N, A, LDA, ILO, IHI, SCALE, INFO)
```

C specification:

```
#include "armpl.h"

void sgebal_(const char *job, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, armpl_int_t *ilo, armpl_int_t *ihi,
             float *scale, armpl_int_t *info, ... );
```

Parameters

JOB Input parameter.

JOB is CHARACTER*1

Specifies the operations to be performed on A: = 'N': none: simply set ILO = 1, IHI = N, SCALE(I) = 1.0 for i = 1,...,N; = 'P': permute only; = 'S': scale only; = 'B': both permute and scale.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the input matrix A. On exit, A is overwritten by the balanced matrix. If JOB = 'N', A is not referenced. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

ILO Output parameter.

ILO is INTEGER

IHI Output parameter.

IHI is INTEGER

ILO and IHI are set to integers such that on exit A(i,j) = 0 if i > j and j = 1,...,ILO-1 or I = IHI+1,...,N. If JOB = 'N' or 'S', ILO = 1 and IHI = N.

SCALE Output parameter.

SCALE is REAL

SCALE is an array, dimension (N). Details of the permutations and scaling factors applied to A. If P(j) is the index of the row and column interchanged with row and column j and D(j) is the scaling factor applied to row and column j, then SCALE(j) = P(j) for j = 1,...,ILO-1 = D(j) for j = ILO,...,IHI = P(j) for j = IHI+1,...,N. The order in which the interchanges are made is N to IHI+1, then 1 to ILO-1.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cgebal*, *dgebal* and *zgebal*. It also exists with a native C interface as *LAPACKE_sgebal*.

4.13.35 sgees

`sgees` computes for an N-by-N real nonsymmetric matrix A, the eigenvalues, the real Schur form T, and, optionally, the matrix of Schur vectors Z. This gives the Schur factorization A = Z*T*(Z^T).

Optionally, it also orders the eigenvalues on the diagonal of the real Schur form so that selected eigenvalues are at the top left. The leading columns of Z then form an orthonormal basis for the invariant subspace corresponding to the selected eigenvalues.

A matrix is in real Schur form if it is upper quasi-triangular with 1-by-1 and 2-by-2 blocks. 2-by-2 blocks will be standardized in the form

```
[  a  b  ]
[  c  a  ]
```

where b*c < 0. The eigenvalues of such a block are a +- sqrt(bc).

Syntax

Fortran specification:

```
use armpl_library

subroutine sgees(JOBVS, SORT, SELECT, N, A, LDA, SDIM, WR, WI, VS, LDVS, WORK,
                 LWORK, BWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgees_(const char *jobvs, const char *sort, ARMPL_SGEES_SELECT select,
            const armpl_int_t *n, float *a, const armpl_int_t *lda,
            armpl_int_t *sdim, float *wr, float *wi, float *vs,
            const armpl_int_t *ldvs, float *work, const armpl_int_t *lwork,
            armpl_int_t *bwork, armpl_int_t *info, ... );
```

Parameters

JOBVS Input parameter.

`JOBVS` is `CHARACTER*1`

= 'N': Schur vectors are not computed; = 'V': Schur vectors are computed.

SORT Input parameter.

`SORT` is `CHARACTER*1`

Specifies whether or not to order the eigenvalues on the diagonal of the Schur form. = 'N': Eigenvalues are not ordered; = 'S': Eigenvalues are ordered (see SELECT).

SELECT Input parameter.

`SELECT` is a `LOGICAL` FUNCTION of two `REAL` arguments

`SELECT` must be declared EXTERNAL in the calling subroutine. If `SORT` = 'S', `SELECT` is used to select eigenvalues to sort to the top left of the Schur form. If `SORT` = 'N', `SELECT` is not referenced. An eigenvalue WR(j)+sqrt(-1)*WI(j) is selected if SELECT(WR(j),WI(j)) is true; i.e., if either one of a complex conjugate pair of eigenvalues is selected, then both complex eigenvalues are selected. Note that a selected complex eigenvalue may no longer satisfy SELECT(WR(j),WI(j)) = .TRUE. after ordering, since ordering may change the value of complex eigenvalues (especially if the eigenvalue is ill-conditioned); in this case `INFO` is set to N+2 (see `INFO` below).

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A. On exit, A has been overwritten by its real Schur form T.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

SDIM Output parameter.

SDIM is INTEGER

If SORT = 'N', SDIM = 0. If SORT = 'S', SDIM = number of eigenvalues (after sorting) for which SELECT is true. (Complex conjugate pairs for which SELECT is true for either eigenvalue count as 2.)

WR Output parameter.

WR is REAL

WR is an array, dimension (N) .

WI Output parameter.

WI is REAL

WI is an array, dimension (N). WR and WI contain the real and imaginary parts, respectively, of the computed eigenvalues in the same order that they appear on the diagonal of the output Schur form T. Complex conjugate pairs of eigenvalues will appear consecutively with the eigenvalue having the positive imaginary part first.

VS Output parameter.

VS is REAL

VS is an array, dimension (LDVS, N). If JOBVS = 'V', VS contains the orthogonal matrix Z of Schur vectors. If JOBVS = 'N', VS is not referenced.

LDVS Input parameter.

LDVS is INTEGER

The leading dimension of the array VS. LDVS >= 1; if JOBVS = 'V', LDVS >= N.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) contains the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,3*N). For good performance, LWORK must generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

BWORK Output parameter.

BWORK is LOGICAL

BWORK is an array, dimension (N). Not referenced if SORT = 'N'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, and i is <= N: the QR algorithm failed to compute all the eigenvalues; elements 1:ILO-1 and i+1:N of WR and WI contain those eigenvalues which have converged; if JOBVS = 'V', VS contains the matrix which reduces A to its partially converged Schur form. = N+1: the eigenvalues could not be reordered because some eigenvalues were too close to separate (the problem is very ill-conditioned); = N+2: after reordering, roundoff changed values of some complex eigenvalues so that leading eigenvalues in the Schur form no longer satisfy SELECT=.TRUE. This could also be caused by underflow due to scaling.

Related Information

For this routine in other precisions, please see *cgees*, *dgees* and *zgees*. It also exists with a native C interface as *LAPACKE_sgees*.

4.13.36 sgeesx

sgeesx computes for an N-by-N real nonsymmetric matrix A, the eigenvalues, the real Schur form T, and, optionally, the matrix of Schur vectors Z. This gives the Schur factorization A = Z*T*(Z^T).

Optionally, it also orders the eigenvalues on the diagonal of the real Schur form so that selected eigenvalues are at the top left; computes a reciprocal condition number for the average of the selected eigenvalues (RCONDE); and computes a reciprocal condition number for the right invariant subspace corresponding to the selected eigenvalues (RCONDV). The leading columns of Z form an orthonormal basis for this invariant subspace.

For further explanation of the reciprocal condition numbers RCONDE and RCONDV, see Section 4.10 of the LAPACK Users' Guide (where these quantities are called s and sep respectively).

A real matrix is in real Schur form if it is upper quasi-triangular with 1-by-1 and 2-by-2 blocks. 2-by-2 blocks will be standardized in the form

```
[  a  b  ]
[  c  a  ]
```

where b*c < 0. The eigenvalues of such a block are a +- sqrt(bc).

Syntax

Fortran specification:

```
use armpl_library

subroutine sgeesx(JOBVS, SORT, SELECT, SENSE, N, A, LDA, SDIM, WR, WI, VS,
                  LDVS, RCONDE, RCONDV, WORK, LWORK, IWORK, LIWORK, BWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void sgeesx_(const char *jobvs, const char *sort, ARMPL_SGEESX_SELECT select,
             const char *sense, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, armpl_int_t *sdim, float *wr, float *wi,
```

(continues on next page)

(continued from previous page)

```
                float *vs, const armpl_int_t *ldvs, float *rconde, float *rcondv,
                float *work, const armpl_int_t *lwork, armpl_int_t *iwork,
                const armpl_int_t *liwork, armpl_int_t *bwork, armpl_int_t *info,
                ... );
```

Parameters

JOBVS Input parameter.

JOBVS is CHARACTER*1

= 'N': Schur vectors are not computed; = 'V': Schur vectors are computed.

SORT Input parameter.

SORT is CHARACTER*1

Specifies whether or not to order the eigenvalues on the diagonal of the Schur form. = 'N': Eigenvalues are not ordered; = 'S': Eigenvalues are ordered (see SELECT).

SELECT Input parameter.

SELECT is a LOGICAL FUNCTION of two REAL arguments

SELECT must be declared EXTERNAL in the calling subroutine. If SORT = 'S', SELECT is used to select eigenvalues to sort to the top left of the Schur form. If SORT = 'N', SELECT is not referenced. An eigenvalue WR(j)+sqrt(-1)*WI(j) is selected if SELECT(WR(j),WI(j)) is true; i.e., if either one of a complex conjugate pair of eigenvalues is selected, then both are. Note that a selected complex eigenvalue may no longer satisfy SELECT(WR(j),WI(j)) = .TRUE. after ordering, since ordering may change the value of complex eigenvalues (especially if the eigenvalue is ill-conditioned); in this case INFO may be set to N+3 (see INFO below).

SENSE Input parameter.

SENSE is CHARACTER*1

Determines which reciprocal condition numbers are computed. = 'N': None are computed; = 'E': Computed for average of selected eigenvalues only; = 'V': Computed for selected right invariant subspace only; = 'B': Computed for both. If SENSE = 'E', 'V' or 'B', SORT must equal 'S'.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A. On exit, A is overwritten by its real Schur form T.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

SDIM Output parameter.

SDIM is INTEGER

If SORT = 'N', SDIM = 0. If SORT = 'S', SDIM = number of eigenvalues (after sorting) for which SELECT is true. (Complex conjugate pairs for which SELECT is true for either eigenvalue count as 2.)

WR Output parameter.

WR is REAL

WR is an array, dimension (N) .

WI Output parameter.

WI is REAL

WI is an array, dimension (N). WR and WI contain the real and imaginary parts, respectively, of the computed eigenvalues, in the same order that they appear on the diagonal of the output Schur form T. Complex conjugate pairs of eigenvalues appear consecutively with the eigenvalue having the positive imaginary part first.

VS Output parameter.

VS is REAL

VS is an array, dimension (LDVS, N). If JOBVS = 'V', VS contains the orthogonal matrix Z of Schur vectors. If JOBVS = 'N', VS is not referenced.

LDVS Input parameter.

LDVS is INTEGER

The leading dimension of the array VS. LDVS >= 1, and if JOBVS = 'V', LDVS >= N.

RCONDE Output parameter.

RCONDE is REAL

If SENSE = 'E' or 'B', RCONDE contains the reciprocal condition number for the average of the selected eigenvalues. Not referenced if SENSE = 'N' or 'V'.

RCONDV Output parameter.

RCONDV is REAL

If SENSE = 'V' or 'B', RCONDV contains the reciprocal condition number for the selected right invariant subspace. Not referenced if SENSE = 'N' or 'E'.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,3*N). Also, if SENSE = 'E' or 'V' or 'B', LWORK >= N+2*SDIM*(N-SDIM), where SDIM is the number of selected eigenvalues computed by this routine. Note that N+2*SDIM*(N-SDIM) <= N+N*N/2. Note also that an error is only returned if LWORK < max(1,3*N), but if SENSE = 'E' or 'V' or 'B' this may not be large enough. For good performance, LWORK must generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates upper bounds on the optimal sizes of the arrays WORK and IWORK, returns these values as the first entries of the WORK and IWORK arrays, and no error messages related to LWORK or LIWORK are issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. LIWORK >= 1; if SENSE = 'V' or 'B', LIWORK >= SDIM*(N-SDIM). Note that SDIM*(N-SDIM) <= N*N/4. Note also that an error is only returned if LIWORK < 1, but if SENSE = 'V' or 'B' this may not be large enough.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates upper bounds on the optimal sizes of the arrays WORK and IWORK, returns these values as the first entries of the WORK and IWORK arrays, and no error messages related to LWORK or LIWORK are issued by XERBLA.

BWORK Output parameter.

BWORK is LOGICAL

BWORK is an array, dimension (N). Not referenced if SORT = 'N'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, and i is <= N: the QR algorithm failed to compute all the eigenvalues; elements 1:ILO-1 and i+1:N of WR and WI contain those eigenvalues which have converged; if JOBVS = 'V', VS contains the transformation which reduces A to its partially converged Schur form. = N+1: the eigenvalues could not be reordered because some eigenvalues were too close to separate (the problem is very ill-conditioned); = N+2: after reordering, roundoff changed values of some complex eigenvalues so that leading eigenvalues in the Schur form no longer satisfy SELECT=.TRUE. This could also be caused by underflow due to scaling.

Related Information

For this routine in other precisions, please see *cgeesx*, *dgeesx* and *zgeesx*. It also exists with a native C interface as *LAPACKE_sgeesx*.

4.13.37 sgeev

sgeev computes for an N-by-N real nonsymmetric matrix A, the eigenvalues and, optionally, the left and/or right eigenvectors.

The right eigenvector v(j) of A satisfies

```
A * v(j) = lambda(j) * v(j)
```

where lambda(j) is its eigenvalue. The left eigenvector u(j) of A satisfies

```
u(j)**H * A = lambda(j) * u(j)**H
```

where $u(j)^H$ denotes the conjugate-transpose of u(j).

The computed eigenvectors are normalized to have Euclidean norm equal to 1 and largest component real.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgeev(JOBVL, JOBVR, N, A, LDA, WR, WI, VL, LDVL, VR, LDVR, WORK,
                 LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgeev_(const char *jobvl, const char *jobvr, const armpl_int_t *n,
            float *a, const armpl_int_t *lda, float *wr, float *wi, float *vl,
            const armpl_int_t *ldvl, float *vr, const armpl_int_t *ldvr,
            float *work, const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

JOBVL Input parameter.

JOBVL is CHARACTER*1

= 'N': left eigenvectors of A are not computed; = 'V': left eigenvectors of A are computed.

JOBVR Input parameter.

JOBVR is CHARACTER*1

= 'N': right eigenvectors of A are not computed; = 'V': right eigenvectors of A are computed.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A. On exit, A has been overwritten.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

WR Output parameter.

WR is REAL

WR is an array, dimension (N) .

WI Output parameter.

WI is REAL

WI is an array, dimension (N). WR and WI contain the real and imaginary parts, respectively, of the computed eigenvalues. Complex conjugate pairs of eigenvalues appear consecutively with the eigenvalue having the positive imaginary part first.

VL Output parameter.

VL is REAL

VL is an array, dimension (LDVL, N). If JOBVL = 'V', the left eigenvectors u(j) are stored one after another in the columns of VL, in the same order as their eigenvalues. If JOBVL = 'N', VL is not referenced. If the j-th eigenvalue is real, then u(j) = VL(:,j), the j-th column of VL. If the j-th and (j+1)-st eigenvalues form a complex conjugate pair, then u(j) = VL(:,j) + i*VL(:,j+1) and u(j+1) = VL(:,j) - i*VL(:,j+1).

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the array VL. LDVL >= 1; if JOBVL = 'V', LDVL >= N.

VR Output parameter.

VR is REAL

VR is an array, dimension (LDVR, N). If JOBVR = 'V', the right eigenvectors v(j) are stored one after another in the columns of VR, in the same order as their eigenvalues. If JOBVR = 'N', VR is not referenced. If the j-th eigenvalue is real, then v(j) = VR(:,j), the j-th column of VR. If the j-th and (j+1)-st eigenvalues form a complex conjugate pair, then v(j) = VR(:,j) + i*VR(:,j+1) and v(j+1) = VR(:,j) - i*VR(:,j+1).

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the array VR. LDVR >= 1; if JOBVR = 'V', LDVR >= N.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,3*N), and if JOBVL = 'V' or JOBVR = 'V', LWORK >= 4*N. For good performance, LWORK must generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, the QR algorithm failed to compute all the eigenvalues, and no eigenvectors have been computed; elements i+1:N of WR and WI contain eigenvalues which have converged.

Related Information

For this routine in other precisions, please see *cgeev*, *dgeev* and *zgeev*. It also exists with a native C interface as *LAPACKE_sgeev*.

4.13.38 sgeevx

sgeevx computes for an N-by-N real nonsymmetric matrix A, the eigenvalues and, optionally, the left and/or right eigenvectors.

Optionally also, it computes a balancing transformation to improve the conditioning of the eigenvalues and eigenvectors (ILO, IHI, SCALE, and ABNRM), reciprocal condition numbers for the eigenvalues (RCONDE), and reciprocal condition numbers for the right eigenvectors (RCONDV).

The right eigenvector v(j) of A satisfies

```
A * v(j) = lambda(j) * v(j)
```

where lambda(j) is its eigenvalue. The left eigenvector u(j) of A satisfies

```
u(j)**H * A = lambda(j) * u(j)**H
```

where $u(j)^H$ denotes the conjugate-transpose of u(j).

The computed eigenvectors are normalized to have Euclidean norm equal to 1 and largest component real.

Balancing a matrix means permuting the rows and columns to make it more nearly upper triangular, and applying a diagonal similarity transformation D * A * D**(-1), where D is a diagonal matrix, to make its rows and columns closer in norm and the condition numbers of its eigenvalues and eigenvectors smaller. The computed reciprocal condition numbers correspond to the balanced matrix. Permuting rows and columns will not change the condition numbers (in exact arithmetic) but diagonal scaling will. For further explanation of balancing, see section 4.10.2 of the LAPACK Users' Guide.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgeevx(BALANC, JOBVL, JOBVR, SENSE, N, A, LDA, WR, WI, VL, LDVL,
                  VR, LDVR, ILO, IHI, SCALE, ABNRM, RCONDE, RCONDV, WORK,
                  LWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgeevx_(const char *balanc, const char *jobvl, const char *jobvr,
             const char *sense, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, float *wr, float *wi, float *vl,
             const armpl_int_t *ldvl, float *vr, const armpl_int_t *ldvr,
             armpl_int_t *ilo, armpl_int_t *ihi, float *scale, float *abnrm,
             float *rconde, float *rcondv, float *work,
             const armpl_int_t *lwork, armpl_int_t *iwork, armpl_int_t *info,
             ... );
```

Parameters

BALANC Input parameter.

BALANC is CHARACTER*1

Indicates how the input matrix should be diagonally scaled and/or permuted to improve the conditioning of its eigenvalues. = 'N': Do not diagonally scale or permute; = 'P': Perform permutations to make the matrix more nearly upper triangular. Do not diagonally scale; = 'S': Diagonally scale the matrix, i.e. replace A by D*A*D**(-1), where D is a diagonal matrix chosen to make the rows and columns of A more equal in norm. Do not permute; = 'B': Both diagonally scale and permute A.

Computed reciprocal condition numbers will be for the matrix after balancing and/or permuting. Permuting does not change condition numbers (in exact arithmetic), but balancing does.

JOBVL Input parameter.

JOBVL is CHARACTER*1

= 'N': left eigenvectors of A are not computed; = 'V': left eigenvectors of A are computed. If SENSE = 'E' or 'B', JOBVL must = 'V'.

JOBVR Input parameter.

JOBVR is CHARACTER*1

= 'N': right eigenvectors of A are not computed; = 'V': right eigenvectors of A are computed. If SENSE = 'E' or 'B', JOBVR must = 'V'.

SENSE Input parameter.

SENSE is CHARACTER*1

Determines which reciprocal condition numbers are computed. = 'N': None are computed; = 'E': Computed for eigenvalues only; = 'V': Computed for right eigenvectors only; = 'B': Computed for eigenvalues and right eigenvectors.

If SENSE = 'E' or 'B', both left and right eigenvectors must also be computed (JOBVL = 'V' and JOBVR = 'V').

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A. On exit, A has been overwritten. If JOBVL = 'V' or JOBVR = 'V', A contains the real Schur form of the balanced version of the input matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

WR Output parameter.

WR is REAL

WR is an array, dimension (N) .

WI Output parameter.

WI is REAL

WI is an array, dimension (N). WR and WI contain the real and imaginary parts, respectively, of the computed eigenvalues. Complex conjugate pairs of eigenvalues will appear consecutively with the eigenvalue having the positive imaginary part first.

VL Output parameter.

VL is REAL

VL is an array, dimension (LDVL, N). If JOBVL = 'V', the left eigenvectors u(j) are stored one after another in the columns of VL, in the same order as their eigenvalues. If JOBVL = 'N', VL is not referenced. If the j-th eigenvalue is real, then u(j) = VL(:,j), the j-th column of VL. If the j-th and (j+1)-st eigenvalues form a complex conjugate pair, then u(j) = VL(:,j) + i*VL(:,j+1) and u(j+1) = VL(:,j) - i*VL(:,j+1).

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the array VL. LDVL >= 1; if JOBVL = 'V', LDVL >= N.

VR Output parameter.

VR is REAL

VR is an array, dimension (LDVR, N). If JOBVR = 'V', the right eigenvectors v(j) are stored one after another in the columns of VR, in the same order as their eigenvalues. If JOBVR = 'N', VR is not referenced. If the j-th eigenvalue is real, then v(j) = VR(:,j), the j-th column of VR. If the j-th and (j+1)-st eigenvalues form a complex conjugate pair, then v(j) = VR(:,j) + i*VR(:,j+1) and v(j+1) = VR(:,j) - i*VR(:,j+1).

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the array VR. LDVR >= 1, and if JOBVR = 'V', LDVR >= N.

ILO Output parameter.

ILO is INTEGER

IHI Output parameter.

IHI is INTEGER

ILO and IHI are integer values determined when A was balanced. The balanced A(i,j) = 0 if I > J and J = 1,. . . ,ILO-1 or I = IHI+1,. . . ,N.

SCALE Output parameter.

SCALE is REAL

SCALE is an array, dimension (N). Details of the permutations and scaling factors applied when balancing A. If P(j) is the index of the row and column interchanged with row and column j, and D(j) is the scaling factor applied to row and column j, then SCALE(J) = P(J), for J = 1,...,ILO-1 = D(J), for J = ILO,...,IHI = P(J) for J = IHI+1,...,N. The order in which the interchanges are made is N to IHI+1, then 1 to ILO-1.

ABNRM Output parameter.

ABNRM is REAL

The one-norm of the balanced matrix (the maximum of the sum of absolute values of elements of any column).

RCONDE Output parameter.

RCONDE is REAL

RCONDE is an array, dimension (N). RCONDE(j) is the reciprocal condition number of the j-th eigenvalue.

RCONDV Output parameter.

RCONDV is REAL

RCONDV is an array, dimension (N). RCONDV(j) is the reciprocal condition number of the j-th right eigenvector.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SENSE = 'N' or 'E', LWORK >= max(1,2*N), and if JOBVL = 'V' or JOBVR = 'V', LWORK >= 3*N. If SENSE = 'V' or 'B', LWORK >= N*(N+6). For good performance, LWORK must generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (2*N-2)

If SENSE = 'N' or 'E', not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, the QR algorithm failed to compute all the eigenvalues, and no eigenvectors or condition numbers have been computed; elements 1:ILO-1 and i+1:N of WR and WI contain eigenvalues which have converged.

Related Information

For this routine in other precisions, please see *cgeevx*, *dgeevx* and *zgeevx*. It also exists with a native C interface as *LAPACKE_sgeevx*.

4.13.39 sgehrd

sgehrd reduces a real general matrix A to upper Hessenberg form H by an orthogonal similarity transformation: $Q^T * A * Q = H$.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgehrd(N, ILO, IHI, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgehrd_(const armpl_int_t *n, const armpl_int_t *ilo,
             const armpl_int_t *ihi, float *a, const armpl_int_t *lda,
             float *tau, float *work, const armpl_int_t *lwork,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

It is assumed that A is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N. ILO and IHI are normally set by a previous call to SGEBAL; otherwise they should be set to 1 and N respectively. See Further Details. 1 <= ILO <= IHI <= N, if N > 0; ILO=1 and IHI=0, if N=0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the N-by-N general matrix to be reduced. On exit, the upper triangle and the first subdiagonal of A are overwritten with the upper Hessenberg matrix H, and the elements below the first subdiagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TAU Output parameter.

TAU is REAL

TAU is an array, dimension (N-1). The scalar factors of the elementary reflectors (see Further Details). Elements 1:ILO-1 and IHI:N-1 of TAU are set to zero.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (LWORK). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= max(1, N). For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cgehrd*, *dgehrd* and *zgehrd*. It also exists with a native C interface as *LAPACKE_sgehrd*.

4.13.40 shsein

shsein uses inverse iteration to find specified right and/or left eigenvectors of a real upper Hessenberg matrix H.

The right eigenvector x and the left eigenvector y of the matrix H corresponding to an eigenvalue w are defined by:

```
H * x = w * x,      y**h * H = w * y**h
```

where y**h denotes the conjugate transpose of the vector y.

Syntax

Fortran specification:

```
use armpl_library

subroutine shsein(SIDE, EIGSRC, INITV, SELECT, N, H, LDH, WR, WI, VL, LDVL,
                  VR, LDVR, MM, M, WORK, IFAILL, IFAILR, INFO)
```

C specification:

```
#include "armpl.h"

void shsein_(const char *side, const char *eigsrc, const char *initv,
             armpl_int_t *select, const armpl_int_t *n, const float *h,
             const armpl_int_t *ldh, float *wr, const float *wi, float *vl,
             const armpl_int_t *ldvl, float *vr, const armpl_int_t *ldvr,
             const armpl_int_t *mm, armpl_int_t *m, float *work,
             armpl_int_t *ifaill, armpl_int_t *ifailr, armpl_int_t *info,
             ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'R': compute right eigenvectors only; = 'L': compute left eigenvectors only; = 'B': compute both right and left eigenvectors.

EIGSRC Input parameter.

EIGSRC is CHARACTER*1

Specifies the source of eigenvalues supplied in (WR, WI): = 'Q': the eigenvalues were found using SHSEQR; thus, if H has zero subdiagonal elements, and so is block-triangular, then the j-th eigenvalue can be assumed to be an eigenvalue of the block containing the j-th row/column. This property allows SHSEIN to perform inverse iteration on just one diagonal block. = 'N': no assumptions are made on the correspondence between eigenvalues and diagonal blocks. In this case, SHSEIN must always perform inverse iteration using the whole matrix H.

INITV Input parameter.

INITV is CHARACTER*1

= 'N': no initial vectors are supplied; = 'U': user-supplied initial vectors are stored in the arrays VL and/or VR.

SELECT Input and output parameter.

SELECT is LOGICAL

SELECT is an array, dimension (N). Specifies the eigenvectors to be computed. To select the real eigenvector corresponding to a real eigenvalue WR(j), SELECT(j) must be set to .TRUE.. To select the complex eigenvector corresponding to a complex eigenvalue (WR(j),WI(j)), with complex conjugate (WR(j+1),WI(j+1)), either SELECT(j) or SELECT(j+1) or both must be set to .TRUE.; then on exit SELECT(j) is .TRUE. and SELECT(j+1) is .FALSE..

N Input parameter.

N is INTEGER

The order of the matrix H. N >= 0.

H Input parameter.

H is REAL

H is an array, dimension (LDH, N). The upper Hessenberg matrix H. If a NaN is detected in H, the routine will return with INFO=-6.

LDH Input parameter.

LDH is INTEGER

The leading dimension of the array H. LDH >= max(1, N).

WR Input and output parameter.

WR is REAL

WR is an array, dimension (N) .

WI Input parameter.

WI is REAL

WI is an array, dimension (N). On entry, the real and imaginary parts of the eigenvalues of H; a complex conjugate pair of eigenvalues must be stored in consecutive elements of WR and WI. On exit, WR may have been altered since close eigenvalues are perturbed slightly in searching for independent eigenvectors.

VL Input and output parameter.

VL is REAL

VL is an array, dimension (LDVL, MM). On entry, if INITV = 'U' and SIDE = 'L' or 'B', VL must contain starting vectors for the inverse iteration for the left eigenvectors; the starting vector for each eigenvector must be in the same column(s) in which the eigenvector will be stored. On exit, if SIDE = 'L' or 'B', the left eigenvectors specified by SELECT will be stored consecutively in the columns of VL, in the same order as their eigenvalues. A complex eigenvector corresponding to a complex eigenvalue is stored in two consecutive columns, the first holding the real part and the second the imaginary part. If SIDE = 'R', VL is not referenced.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the array VL. LDVL >= max(1, N) if SIDE = 'L' or 'B'; LDVL >= 1 otherwise.

VR Input and output parameter.

VR is REAL

VR is an array, dimension (LDVR, MM). On entry, if INITV = 'U' and SIDE = 'R' or 'B', VR must contain starting vectors for the inverse iteration for the right eigenvectors; the starting vector for each eigenvector must be in the same column(s) in which the eigenvector will be stored. On exit, if SIDE = 'R' or 'B', the right eigenvectors specified by SELECT will be stored consecutively in the columns of VR, in the same order as their eigenvalues. A complex eigenvector corresponding to a complex eigenvalue is stored in two consecutive columns, the first holding the real part and the second the imaginary part. If SIDE = 'L', VR is not referenced.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the array VR. LDVR >= max(1, N) if SIDE = 'R' or 'B'; LDVR >= 1 otherwise.

MM Input parameter.

MM is INTEGER

The number of columns in the arrays VL and/or VR. MM >= M.

M Output parameter.

M is INTEGER

The number of columns in the arrays VL and/or VR required to store the eigenvectors; each selected real eigenvector occupies one column and each selected complex eigenvector occupies two columns.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension ((N+2)*N) .

IFAILL Output parameter.

IFAILL is INTEGER array, dimension (MM)

If SIDE = 'L' or 'B', IFAILL(i) = j > 0 if the left eigenvector in the i-th column of VL (corresponding to the eigenvalue w(j)) failed to converge; IFAILL(i) = 0 if the eigenvector converged satisfactorily. If the i-th and (i+1)th columns of VL hold a complex eigenvector, then IFAILL(i) and IFAILL(i+1) are set to the same value. If SIDE = 'R', IFAILL is not referenced.

IFAILR Output parameter.

IFAILR is INTEGER array, dimension (MM)

If SIDE = 'R' or 'B', IFAILR(i) = j > 0 if the right eigenvector in the i-th column of VR (corresponding to the eigenvalue w(j)) failed to converge; IFAILR(i) = 0 if the eigenvector converged satisfactorily. If the i-th and (i+1)th columns of VR hold a complex eigenvector, then IFAILR(i) and IFAILR(i+1) are set to the same value. If SIDE = 'L', IFAILR is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, i is the number of eigenvectors which failed to converge; see IFAILL and IFAILR for further details.

Related Information

For this routine in other precisions, please see *chsein*, *dhsein* and *zhsein*. It also exists with a native C interface as *LAPACKE_shsein*.

4.13.41 shseqr

```
SHSEQR computes the eigenvalues of a Hessenberg matrix H
   and, optionally, the matrices T and Z from the Schur decomposition
   H = Z T Z**T, where T is an upper quasi-triangular matrix (the
   Schur form), and Z is the orthogonal matrix of Schur vectors.

   Optionally Z may be postmultiplied into an input orthogonal
   matrix Q so that this routine can give the Schur factorization
   of a matrix A which has been reduced to the Hessenberg form H
   by the orthogonal matrix Q:  A = Q*H*Q**T = (QZ)*T*(QZ)**T.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine shseqr(JOB, COMPZ, N, ILO, IHI, H, LDH, WR, WI, Z, LDZ, WORK,
                  LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void shseqr_(const char *job, const char *compz, const armpl_int_t *n,
             const armpl_int_t *ilo, const armpl_int_t *ihi, float *h,
             const armpl_int_t *ldh, float *wr, float *wi, float *z,
             const armpl_int_t *ldz, float *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

JOB Input parameter.

JOB is CHARACTER*1

= ‘E’: compute eigenvalues only; = ‘S’: compute eigenvalues and the Schur form T.

COMPZ Input parameter.

COMPZ is CHARACTER*1

= ‘N’: no Schur vectors are computed; = ‘I’: Z is initialized to the unit matrix and the matrix Z of Schur vectors of H is returned; = ‘V’: Z must contain an orthogonal matrix Q on entry, and the product Q* Z is returned.

N Input parameter.

N is INTEGER

The order of the matrix H. N .GE. 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

It is assumed that H is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N. ILO and IHI are normally set by a previous call to SGEBAL, and then passed to ZGEHRD when the matrix output by SGEBAL is reduced to Hessenberg form. Otherwise ILO and IHI should be set to 1 and N respectively. If N.GT.0, then 1.LE.ILO.LE.IHI.LE.N. If N = 0, then ILO = 1 and IHI = 0.

H Input and output parameter.

H is REAL

H is an array, dimension (LDH, N). On entry, the upper Hessenberg matrix H. On exit, if INFO = 0 and JOB = 'S', then H contains the upper quasi-triangular matrix T from the Schur decomposition (the Schur form); 2-by-2 diagonal blocks (corresponding to complex conjugate pairs of eigenvalues) are returned in standard form, with H(i,i) = H(i+1,i+1) and H(i+1,i)*H(i,i+1).LT.0. If INFO = 0 and JOB = 'E', the contents of H are unspecified on exit. (The output value of H when INFO.GT.0 is given under the description of INFO below.)

Unlike earlier versions of SHSEQR, this subroutine may explicitly H(i,j) = 0 for i.GT.j and j = 1, 2, ... ILO-1 or j = IHI+1, IHI+2, ... N.

LDH Input parameter.

LDH is INTEGER

The leading dimension of the array H. LDH .GE. max(1, N).

WR Output parameter.

WR is REAL

WR is an array, dimension (N) .

WI Output parameter.

WI is REAL

WI is an array, dimension (N). The real and imaginary parts, respectively, of the computed eigenvalues. If two eigenvalues are computed as a complex conjugate pair, they are stored in consecutive elements of WR and WI, say the i-th and (i+1)th, with WI(i) .GT. 0 and WI(i+1) .LT. 0. If JOB = 'S', the eigenvalues are stored in the same order as on the diagonal of the Schur form returned in H, with WR(i) = H(i,i) and, if H(i:i+1,i:i+1) is a 2-by-2 diagonal block, WI(i) = sqrt(-H(i+1,i)*H(i,i+1)) and WI(i+1) = -WI(i).

Z Input and output parameter.

Z is REAL

Z is an array, dimension (LDZ, N). If COMPZ = 'N', Z is not referenced. If COMPZ = 'I', on entry Z need not be set and on exit, if INFO = 0, Z contains the orthogonal matrix Z of the Schur vectors of H. If COMPZ = 'V', on entry Z must contain an N-by-N matrix Q, which is assumed to be equal to the unit matrix except for the submatrix Z(ILO:IHI,ILO:IHI). On exit, if INFO = 0, Z contains Q*Z. Normally Q is the orthogonal matrix generated by SORGHR after the call to SGEHRD which formed the Hessenberg matrix H. (The output value of Z when INFO.GT.0 is given under the description of INFO below.)

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. if COMPZ = 'I' or COMPZ = 'V', then LDZ.GE.MAX(1, N). Otherwize, LDZ.GE.1.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (LWORK). On exit, if INFO = 0, WORK(1) returns an estimate of the optimal value for LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK .GE. max(1, N) is sufficient and delivers very good and sometimes optimal performance. However, LWORK as large as 11*N may be required for optimal performance. A workspace query is recommended to determine the optimal workspace size.

If LWORK = -1, then SHSEQR does a workspace query. In this case, SHSEQR checks the input parameters and estimates the optimal workspace size for the given values of N, ILO and IHI. The estimate is returned in WORK(1). No error message related to LWORK is issued by XERBLA. Neither H nor Z are accessed.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit .LT. 0: if INFO = -i, the i-th argument had an illegal value .GT. 0: if INFO = i, SHSEQR failed to compute all of the eigenvalues. Elements 1:ilo-1 and i+1:n of WR and WI contain those eigenvalues which have been successfully computed. (Failures are rare.)

If INFO .GT. 0 and JOB = 'E', then on exit, the remaining unconverged eigenvalues are the eigen- values of the upper Hessenberg matrix rows and columns ILO through INFO of the final, output value of H.

If INFO .GT. 0 and JOB = 'S', then on exit

(*) (initial value of H)*U = U*(final value of H)

where U is an orthogonal matrix. The final value of H is upper Hessenberg and quasi-triangular in rows and columns INFO+1 through IHI.

If INFO .GT. 0 and COMPZ = 'V', then on exit

(final value of Z) = (initial value of Z)*U

where U is the orthogonal matrix in (*) (regard- less of the value of JOB.)

If INFO .GT. 0 and COMPZ = 'I', then on exit (final value of Z) = U where U is the orthogonal matrix in (*) (regard- less of the value of JOB.)

If INFO .GT. 0 and COMPZ = 'N', then Z is not accessed.

Related Information

For this routine in other precisions, please see *chseqr*, *dhseqr* and *zhseqr*. It also exists with a native C interface as *LAPACKE_shseqr*.

4.13.42 sorghr

sorghr generates a real orthogonal matrix Q which is defined as the product of IHI-ILO elementary reflectors of order N, as returned by SGEHRD:

Q = H(ilo) H(ilo+1) . . . H(ihi-1).

Syntax

Fortran specification:

```
use armpl_library

subroutine sorghr(N, ILO, IHI, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sorghr_(const armpl_int_t *n, const armpl_int_t *ilo,
             const armpl_int_t *ihi, float *a, const armpl_int_t *lda,
             const float *tau, float *work, const armpl_int_t *lwork,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix Q. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

ILO and IHI must have the same values as in the previous call of SGEHRD. Q is equal to the unit matrix except in the submatrix Q(ilo+1:ihi,ilo+1:ihi). 1 <= ILO <= IHI <= N, if N > 0; ILO=1 and IHI=0, if N=0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the vectors which define the elementary reflectors, as returned by SGEHRD. On exit, the N-by-N orthogonal matrix Q.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TAU Input parameter.

TAU is REAL

TAU is an array, dimension (N-1). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by SGEHRD.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= IHI-ILO. For optimum performance LWORK >= (IHI-ILO)*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dorghr*. It also exists with a native C interface as *LAPACKE_sorghr*.

4.13.43 sormhr

`sormhr` overwrites the general real M-by-N matrix C with

```
SIDE = 'L'      SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'T': Q^T * C C * Q^T

where Q is a real orthogonal matrix of order nq, with nq = m if SIDE = 'L' and nq = n if SIDE = 'R'. Q is defined as the product of IHI-ILO elementary reflectors, as returned by SGEHRD:

Q = H(ilo) H(ilo+1) . . . H(ihi-1).

Syntax

Fortran specification:

```
use armpl_library

subroutine sormhr(SIDE, TRANS, M, N, ILO, IHI, A, LDA, TAU, C, LDC, WORK,
                  LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sormhr_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *ilo,
             const armpl_int_t *ihi, const float *a, const armpl_int_t *lda,
             const float *tau, float *c, const armpl_int_t *ldc, float *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'T': Transpose, apply Q^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

ILO and IHI must have the same values as in the previous call of SGEHRD. Q is equal to the unit matrix except in the submatrix Q(ilo+1:ihi,ilo+1:ihi). If SIDE = 'L', then 1 <= ILO <= IHI <= M, if M > 0, and ILO = 1 and IHI = 0, if M = 0; if SIDE = 'R', then 1 <= ILO <= IHI <= N, if N > 0, and ILO = 1 and IHI = 0, if N = 0.

A Input parameter.

A is REAL

A is an array, dimension. (LDA, M) if SIDE = 'L' (LDA, N) if SIDE = 'R' The vectors which define the elementary reflectors, as returned by SGEHRD.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M) if SIDE = 'L'; LDA >= max(1, N) if SIDE = 'R'.

TAU Input parameter.

TAU is REAL

TAU is an array, dimension. (M-1) if SIDE = 'L' (N-1) if SIDE = 'R' TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by SGEHRD.

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, N); if SIDE = 'R', LWORK >= max(1, M). For optimum performance LWORK >= N*NB if SIDE = 'L', and LWORK >= M*NB if SIDE = 'R', where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dormhr*. It also exists with a native C interface as *LAPACKE_sormhr*.

4.13.44 strevc

strevc computes some or all of the right and/or left eigenvectors of a real upper quasi-triangular matrix T. Matrices of this type are produced by the Schur factorization of a real general matrix: A = Q*T*Q^T , as computed by SHSEQR.

The right eigenvector x and the left eigenvector y of T corresponding to an eigenvalue w are defined by:

```
T*x = w*x,      (y**H)*T = w*(y**H)
```

where y^H denotes the conjugate transpose of y. The eigenvalues are not input to this routine, but are read directly from the diagonal blocks of T.

This routine returns the matrices X and/or Y of right and left eigenvectors of T, or the products Q*X and/or Q*Y, where Q is an input matrix. If Q is the orthogonal factor that reduces a matrix A to Schur form T, then Q*X and Q*Y are the matrices of right and left eigenvectors of A.

Syntax

Fortran specification:

```
use armpl_library

subroutine strevc(SIDE, HOWMNY, SELECT, N, T, LDT, VL, LDVL, VR, LDVR, MM, M,
                  WORK, INFO)
```

C specification:

```
#include "armpl.h"

void strevc_(const char *side, const char *howmny, armpl_int_t *select,
             const armpl_int_t *n, const float *t, const armpl_int_t *ldt,
             float *vl, const armpl_int_t *ldvl, float *vr,
             const armpl_int_t *ldvr, const armpl_int_t *mm, armpl_int_t *m,
             float *work, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'R': compute right eigenvectors only; = 'L': compute left eigenvectors only; = 'B': compute both right and left eigenvectors.

HOWMNY Input parameter.

HOWMNY is CHARACTER*1

= 'A': compute all right and/or left eigenvectors; = 'B': compute all right and/or left eigenvectors, backtransformed by the matrices in VR and/or VL; = 'S': compute selected right and/or left eigenvectors, as indicated by the logical array SELECT.

SELECT Input and output parameter.

SELECT is LOGICAL

SELECT is an array, dimension (N). If HOWMNY = 'S', SELECT specifies the eigenvectors to be computed. If w(j) is a real eigenvalue, the corresponding real eigenvector is computed if SELECT(j) is .TRUE.. If w(j) and w(j+1) are the real and imaginary parts of a complex eigenvalue, the corresponding complex eigenvector is computed if either SELECT(j) or SELECT(j+1) is .TRUE., and on exit SELECT(j) is set to .TRUE. and SELECT(j+1) is set to .FALSE.. Not referenced if HOWMNY = 'A' or 'B'.

N Input parameter.

N is INTEGER

The order of the matrix T. N >= 0.

T Input parameter.

T is REAL

T is an array, dimension (LDT, N). The upper quasi-triangular matrix T in Schur canonical form.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

VL Input and output parameter.

VL is REAL

VL is an array, dimension (LDVL, MM). On entry, if SIDE = 'L' or 'B' and HOWMNY = 'B', VL must contain an N-by-N matrix Q (usually the orthogonal matrix Q of Schur vectors returned by SHSEQR). On exit, if SIDE = 'L' or 'B', VL contains: if HOWMNY = 'A', the matrix Y of left eigenvectors of T; if HOWMNY = 'B', the matrix Q*Y; if HOWMNY = 'S', the left eigenvectors of T specified by SELECT, stored consecutively in the columns of VL, in the same order as their eigenvalues. A complex eigenvector corresponding to a complex eigenvalue is stored in two consecutive columns, the first holding the real part, and the second the imaginary part. Not referenced if SIDE = 'R'.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the array VL. LDVL >= 1, and if SIDE = 'L' or 'B', LDVL >= N.

VR Input and output parameter.

VR is REAL

VR is an array, dimension (LDVR, MM). On entry, if SIDE = 'R' or 'B' and HOWMNY = 'B', VR must contain an N-by-N matrix Q (usually the orthogonal matrix Q of Schur vectors returned by SHSEQR). On exit, if SIDE = 'R' or 'B', VR contains: if HOWMNY = 'A', the matrix X of right eigenvectors of T; if HOWMNY = 'B', the matrix Q*X; if HOWMNY = 'S', the right eigenvectors of T specified by SELECT, stored consecutively in the columns of VR, in the same order as their eigenvalues. A complex eigenvector corresponding to a complex eigenvalue is stored in two consecutive columns, the first holding the real part and the second the imaginary part. Not referenced if SIDE = 'L'.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the array VR. LDVR >= 1, and if SIDE = 'R' or 'B', LDVR >= N.

MM Input parameter.

MM is INTEGER

The number of columns in the arrays VL and/or VR. MM >= M.

M Output parameter.

M is INTEGER

The number of columns in the arrays VL and/or VR actually used to store the eigenvectors. If HOWMNY = 'A' or 'B', M is set to N. Each selected real eigenvector occupies one column and each selected complex eigenvector occupies two columns.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (3*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctrevc*, *dtrevc* and *ztrevc*. It also exists with a native C interface as *LAPACKE_strevc*.

4.13.45 strexc

strexc reorders the real Schur factorization of a real matrix A = Q*T*Q^T , so that the diagonal block of T with row index IFST is moved to row ILST.

The real Schur form T is reordered by an orthogonal similarity transformation Z^T *T*Z, and optionally the matrix Q of Schur vectors is updated by postmultiplying it with Z.

T must be in Schur canonical form (as returned by SHSEQR), that is, block upper triangular with 1-by-1 and 2-by-2 diagonal blocks; each 2-by-2 diagonal block has its diagonal elements equal and its off-diagonal elements of opposite sign.

Syntax

Fortran specification:

```
use armpl_library

subroutine strexc(COMPQ, N, T, LDT, Q, LDQ, IFST, ILST, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void strexc_(const char *compq, const armpl_int_t *n, float *t,
             const armpl_int_t *ldt, float *q, const armpl_int_t *ldq,
             armpl_int_t *ifst, armpl_int_t *ilst, float *work,
             armpl_int_t *info, ... );
```

Parameters

COMPQ Input parameter.

COMPQ is CHARACTER*1

= 'V': update the matrix Q of Schur vectors; = 'N': do not update Q.

N Input parameter.

N is INTEGER

The order of the matrix T. N >= 0. If N == 0 arguments ILST and IFST may be any value.

T Input and output parameter.

T is REAL

T is an array, dimension (LDT, N). On entry, the upper quasi-triangular matrix T, in Schur Schur canonical form. On exit, the reordered upper quasi-triangular matrix, again in Schur canonical form.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

Q Input and output parameter.

Q is REAL

Q is an array, dimension (LDQ, N). On entry, if COMPQ = 'V', the matrix Q of Schur vectors. On exit, if COMPQ = 'V', Q has been postmultiplied by the orthogonal transformation matrix Z which reorders T. If COMPQ = 'N', Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= 1, and if COMPQ = 'V', LDQ >= max(1, N).

IFST Input and output parameter.

IFST is INTEGER

ILST Input and output parameter.

ILST is INTEGER

Specify the reordering of the diagonal blocks of T. The block with row index IFST is moved to row ILST, by a sequence of transpositions between adjacent blocks. On exit, if IFST pointed on entry to the second row of a 2-by-2 block, it is changed to point to the first row; ILST always points to the first row of the block in its final position (which may differ from its input value by +1 or -1). 1 <= IFST <= N; 1 <= ILST <= N.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value = 1: two adjacent blocks were too close to swap (the problem is very ill-conditioned); T may have been partially reordered, and ILST points to the first row of the current position of the block being moved.

Related Information

For this routine in other precisions, please see *ctrexc*, *dtrexc* and *ztrexc*. It also exists with a native C interface as *LAPACKE_strexc*.

4.13.46 strsen

strsen reorders the real Schur factorization of a real matrix A = Q*T*Q^T , so that a selected cluster of eigenvalues appears in the leading diagonal blocks of the upper quasi-triangular matrix T, and the leading columns of Q form an orthonormal basis of the corresponding right invariant subspace.

Optionally the routine computes the reciprocal condition numbers of the cluster of eigenvalues and/or the invariant subspace.

T must be in Schur canonical form (as returned by SHSEQR), that is, block upper triangular with 1-by-1 and 2-by-2 diagonal blocks; each 2-by-2 diagonal block has its diagonal elements equal and its off-diagonal elements of opposite sign.

Syntax

Fortran specification:

```
use armpl_library

subroutine strsen(JOB, COMPQ, SELECT, N, T, LDT, Q, LDQ, WR, WI, M, S, SEP,
                  WORK, LWORK, IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void strsen_(const char *job, const char *compq, const armpl_int_t *select,
             const armpl_int_t *n, float *t, const armpl_int_t *ldt, float *q,
             const armpl_int_t *ldq, float *wr, float *wi, armpl_int_t *m,
             float *s, float *sep, float *work, const armpl_int_t *lwork,
             armpl_int_t *iwork, const armpl_int_t *liwork, armpl_int_t *info,
             ... );
```

Parameters

JOB Input parameter.

JOB is CHARACTER*1

Specifies whether condition numbers are required for the cluster of eigenvalues (S) or the invariant subspace (SEP): = 'N': none; = 'E': for eigenvalues only (S); = 'V': for invariant subspace only (SEP); = 'B': for both eigenvalues and invariant subspace (S and SEP).

COMPQ Input parameter.

COMPQ is CHARACTER*1

= 'V': update the matrix Q of Schur vectors; = 'N': do not update Q.

SELECT Input parameter.

SELECT is LOGICAL

SELECT is an array, dimension (N). SELECT specifies the eigenvalues in the selected cluster. To select a real eigenvalue w(j), SELECT(j) must be set to .TRUE.. To select a complex conjugate pair of eigenvalues w(j) and w(j+1), corresponding to a 2-by-2 diagonal block, either SELECT(j) or SELECT(j+1) or both must be set to .TRUE.; a complex conjugate pair of eigenvalues must be either both included in the cluster or both excluded.

N Input parameter.

N is INTEGER

The order of the matrix T. N >= 0.

T Input and output parameter.

T is REAL

T is an array, dimension (LDT, N). On entry, the upper quasi-triangular matrix T, in Schur canonical form. On exit, T is overwritten by the reordered matrix T, again in Schur canonical form, with the selected eigenvalues in the leading diagonal blocks.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

Q Input and output parameter.

Q is REAL

Q is an array, dimension (LDQ, N). On entry, if COMPQ = 'V', the matrix Q of Schur vectors. On exit, if COMPQ = 'V', Q has been postmultiplied by the orthogonal transformation matrix which reorders T; the leading M columns of Q form an orthonormal basis for the specified invariant subspace. If COMPQ = 'N', Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= 1; and if COMPQ = 'V', LDQ >= N.

WR Output parameter.

WR is REAL

WR is an array, dimension (N) .

WI Output parameter.

WI is REAL

WI is an array, dimension (N). The real and imaginary parts, respectively, of the reordered eigenvalues of T. The eigenvalues are stored in the same order as on the diagonal of T, with WR(i) = T(i,i) and, if T(i:i+1,i:i+1) is a 2-by-2 diagonal block, WI(i) > 0 and WI(i+1) = -WI(i). Note that if a complex eigenvalue is sufficiently ill-conditioned, then its value may differ significantly from its value before reordering.

M Output parameter.

M is INTEGER

The dimension of the specified invariant subspace. 0 < = M <= N.

S Output parameter.

S is REAL

If JOB = 'E' or 'B', S is a lower bound on the reciprocal condition number for the selected cluster of eigenvalues. S cannot underestimate the true reciprocal condition number by more than a factor of sqrt(N). If M = 0 or N, S = 1. If JOB = 'N' or 'V', S is not referenced.

SEP Output parameter.

SEP is REAL

If JOB = 'V' or 'B', SEP is the estimated reciprocal condition number of the specified invariant subspace. If M = 0 or N, SEP = norm(T). If JOB = 'N' or 'E', SEP is not referenced.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If JOB = 'N', LWORK >= max(1, N); if JOB = 'E', LWORK >= max(1,M*(N-M)); if JOB = 'V' or 'B', LWORK >= max(1,2*M*(N-M)).

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. If JOB = 'N' or 'E', LIWORK >= 1; if JOB = 'V' or 'B', LIWORK >= max(1,M*(N-M)).

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the IWORK array, returns this value as the first entry of the IWORK array, and no error message related to LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value = 1: reordering of T failed because some eigenvalues are too close to separate (the problem is very ill-conditioned); T may have been partially reordered, and WR and WI contain the eigenvalues in the same order as in T; S and SEP (if requested) are set to zero.

Related Information

For this routine in other precisions, please see *ctrsen*, *dtrsen* and *ztrsen*. It also exists with a native C interface as *LAPACKE_strsen*.

4.13.47 strsna

strsna estimates reciprocal condition numbers for specified eigenvalues and/or right eigenvectors of a real upper quasi-triangular matrix T (or of any matrix $Q*T*Q^T$ with Q orthogonal).

T must be in Schur canonical form (as returned by SHSEQR), that is, block upper triangular with 1-by-1 and 2-by-2 diagonal blocks; each 2-by-2 diagonal block has its diagonal elements equal and its off-diagonal elements of opposite sign.

Syntax

Fortran specification:

```
use armpl_library

subroutine strsna(JOB, HOWMNY, SELECT, N, T, LDT, VL, LDVL, VR, LDVR, S, SEP,
                  MM, M, WORK, LDWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void strsna_(const char *job, const char *howmny, const armpl_int_t *select,
             const armpl_int_t *n, const float *t, const armpl_int_t *ldt,
             const float *vl, const armpl_int_t *ldvl, const float *vr,
             const armpl_int_t *ldvr, float *s, float *sep,
             const armpl_int_t *mm, armpl_int_t *m, float *work,
             const armpl_int_t *ldwork, armpl_int_t *iwork, armpl_int_t *info,
             ... );
```

Parameters

JOB Input parameter.

JOB is CHARACTER*1

Specifies whether condition numbers are required for eigenvalues (S) or eigenvectors (SEP): = 'E': for eigenvalues only (S); = 'V': for eigenvectors only (SEP); = 'B': for both eigenvalues and eigenvectors (S and SEP).

HOWMNY Input parameter.

HOWMNY is CHARACTER*1

= 'A': compute condition numbers for all eigenpairs; = 'S': compute condition numbers for selected eigenpairs specified by the array SELECT.

SELECT Input parameter.

SELECT is LOGICAL

SELECT is an array, dimension (N). If HOWMNY = 'S', SELECT specifies the eigenpairs for which condition numbers are required. To select condition numbers for the eigenpair corresponding to a real eigenvalue w(j), SELECT(j) must be set to .TRUE.. To select condition numbers corresponding to a complex conjugate pair of eigenvalues w(j) and w(j+1), either SELECT(j) or SELECT(j+1) or both, must be set to .TRUE.. If HOWMNY = 'A', SELECT is not referenced.

N Input parameter.

N is INTEGER

The order of the matrix T. N >= 0.

T Input parameter.

T is REAL

T is an array, dimension (LDT, N). The upper quasi-triangular matrix T, in Schur canonical form.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

VL Input parameter.

VL is REAL

VL is an array, dimension (LDVL, M). If JOB = 'E' or 'B', VL must contain left eigenvectors of T (or of any Q*T*Q^T with Q orthogonal), corresponding to the eigenpairs specified by HOWMNY and SELECT. The eigenvectors must be stored in consecutive columns of VL, as returned by SHSEIN or STREVC. If JOB = 'V', VL is not referenced.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the array VL. LDVL >= 1; and if JOB = 'E' or 'B', LDVL >= N.

VR Input parameter.

VR is REAL

VR is an array, dimension (LDVR, M). If JOB = 'E' or 'B', VR must contain right eigenvectors of T (or of any Q*T*Q^T with Q orthogonal), corresponding to the eigenpairs specified by HOWMNY and SELECT. The eigenvectors must be stored in consecutive columns of VR, as returned by SHSEIN or STREVC. If JOB = 'V', VR is not referenced.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the array VR. LDVR >= 1; and if JOB = 'E' or 'B', LDVR >= N.

S Output parameter.

S is REAL

S is an array, dimension (MM). If JOB = 'E' or 'B', the reciprocal condition numbers of the selected eigenvalues, stored in consecutive elements of the array. For a complex conjugate pair of eigenvalues two consecutive elements of S are set to the same value. Thus S(j), SEP(j), and the j-th columns of VL and VR all correspond to the same eigenpair (but not in general the j-th eigenpair, unless all eigenpairs are selected). If JOB = 'V', S is not referenced.

SEP Output parameter.

SEP is REAL

SEP is an array, dimension (MM). If JOB = 'V' or 'B', the estimated reciprocal condition numbers of the selected eigenvectors, stored in consecutive elements of the array. For a complex eigenvector two consecutive elements of SEP are set to the same value. If the eigenvalues cannot be reordered to compute SEP(j), SEP(j) is set to 0; this can only occur when the true value would be very small anyway. If JOB = 'E', SEP is not referenced.

MM Input parameter.

MM is INTEGER

The number of elements in the arrays S (if JOB = 'E' or 'B') and/or SEP (if JOB = 'V' or 'B'). MM >= M.

M Output parameter.

M is INTEGER

The number of elements of the arrays S and/or SEP actually used to store the estimated condition numbers. If HOWMNY = 'A', M is set to N.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (LDWORK,N+6). If JOB = 'E', WORK is not referenced.

LDWORK Input parameter.

LDWORK is INTEGER

The leading dimension of the array WORK. LDWORK >= 1; and if JOB = 'V' or 'B', LDWORK >= N.

IWORK Output parameter.

IWORK is INTEGER array, dimension (2*(N-1))

If JOB = 'E', IWORK is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctrsna*, *dtrsna* and *ztrsna*. It also exists with a native C interface as *LAPACKE_strsna*.

4.13.48 strsyl

`strsyl` solves the real Sylvester matrix equation:

```
op(A)*X + X*op(B) = scale*C or
op(A)*X - X*op(B) = scale*C,
```

where op(A) = A or A^T , and A and B are both upper quasi- triangular. A is M-by-M and B is N-by-N; the right hand side C and the solution X are M-by-N; and scale is an output scale factor, set <= 1 to avoid overflow in X.

A and B must be in Schur canonical form (as returned by SHSEQR), that is, block upper triangular with 1-by-1 and 2-by-2 diagonal blocks; each 2-by-2 diagonal block has its diagonal elements equal and its off-diagonal elements of opposite sign.

Syntax

Fortran specification:

```
use armpl_library

subroutine strsyl(TRANA, TRANB, ISGN, M, N, A, LDA, B, LDB, C, LDC, SCALE,
                  INFO)
```

C specification:

```
#include "armpl.h"

void strsyl_(const char *trana, const char *tranb, const armpl_int_t *isgn,
             const armpl_int_t *m, const armpl_int_t *n, const float *a,
             const armpl_int_t *lda, const float *b, const armpl_int_t *ldb,
             float *c, const armpl_int_t *ldc, float *scale,
             armpl_int_t *info, ... );
```

Parameters

TRANA Input parameter.

TRANA is CHARACTER*1

Specifies the option op(A): = 'N': op(A) = A (No transpose) = 'T': op(A) = A^T (Transpose) = 'C': op(A) = A^H (Conjugate transpose = Transpose)

TRANB Input parameter.

TRANB is CHARACTER*1

Specifies the option op(B): = 'N': op(B) = B (No transpose) = 'T': op(B) = B^T (Transpose) = 'C': op(B) = B^H (Conjugate transpose = Transpose)

ISGN Input parameter.

ISGN is INTEGER

Specifies the sign in the equation: = +1: solve op(A)*X + X*op(B) = scale*C = -1: solve op(A)*X - X*op(B) = scale* C

M Input parameter.

M is INTEGER

The order of the matrix A, and the number of rows in the matrices X and C. M >= 0.

N Input parameter.

N is INTEGER

The order of the matrix B, and the number of columns in the matrices X and C. N >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, M). The upper quasi-triangular matrix A, in Schur canonical form.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input parameter.

B is REAL

B is an array, dimension (LDB, N). The upper quasi-triangular matrix B, in Schur canonical form.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, N). On entry, the M-by-N right hand side matrix C. On exit, C is overwritten by the solution matrix X.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M)

SCALE Output parameter.

SCALE is REAL

The scale factor, scale, set <= 1 to avoid overflow in X.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value = 1: A and B have common or very close eigenvalues; perturbed values were used to solve the equation (but the matrices A and B are unchanged).

Related Information

For this routine in other precisions, please see *ctrsyl*, *dtrsyl* and *ztrsyl*. It also exists with a native C interface as *LAPACKE_strsyl*.

4.13.49 zgebak

zgebak forms the right or left eigenvectors of a complex general matrix by backward transformation on the computed eigenvectors of the balanced matrix output by ZGEBAL.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgebak(JOB, SIDE, N, ILO, IHI, SCALE, M, V, LDV, INFO)
```

C specification:

```
#include "armpl.h"

void zgebak_(const char *job, const char *side, const armpl_int_t *n,
             const armpl_int_t *ilo, const armpl_int_t *ihi,
             const double *scale, const armpl_int_t *m,
             armpl_doublecomplex_t *v, const armpl_int_t *ldv,
             armpl_int_t *info, ... );
```

Parameters

JOB Input parameter.

JOB is CHARACTER*1

Specifies the type of backward transformation required: = 'N', do nothing, return immediately; = 'P', do backward transformation for permutation only; = 'S', do backward transformation for scaling only; = 'B', do backward transformations for both permutation and scaling. JOB must be the same as the argument JOB supplied to ZGEBAL.

SIDE Input parameter.

SIDE is CHARACTER*1

= 'R': V contains right eigenvectors; = 'L': V contains left eigenvectors.

N Input parameter.

N is INTEGER

The number of rows of the matrix V. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

The integers ILO and IHI determined by ZGEBAL. 1 <= ILO <= IHI <= N, if N > 0; ILO=1 and IHI=0, if N=0.

SCALE Input parameter.

SCALE is DOUBLE PRECISION

SCALE is an array, dimension (N). Details of the permutation and scaling factors, as returned by ZGEBAL.

M Input parameter.

M is INTEGER

The number of columns of the matrix V. M >= 0.

V Input and output parameter.

V is COMPLEX*16

V is an array, dimension (LDV, M). On entry, the matrix of right or left eigenvectors to be transformed, as returned by ZHSEIN or ZTREVC. On exit, V is overwritten by the transformed eigenvectors.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. LDV >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cgebak*, *dgebak* and *sgebak*. It also exists with a native C interface as *LAPACKE_zgebak*.

4.13.50 zgebal

zgebal balances a general complex matrix A. This involves, first, permuting A by a similarity transformation to isolate eigenvalues in the first 1 to ILO-1 and last IHI+1 to N elements on the diagonal; and second, applying a diagonal similarity transformation to rows and columns ILO to IHI to make the rows and columns as close in norm as possible. Both steps are optional.

Balancing may reduce the 1-norm of the matrix, and improve the accuracy of the computed eigenvalues and/or eigenvectors.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgebal(JOB, N, A, LDA, ILO, IHI, SCALE, INFO)
```

C specification:

```
#include "armpl.h"

void zgebal_(const char *job, const armpl_int_t *n, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *ilo, armpl_int_t *ihi,
             double *scale, armpl_int_t *info, ... );
```

Parameters

JOB Input parameter.

JOB is CHARACTER*1

Specifies the operations to be performed on A: = 'N': none: simply set ILO = 1, IHI = N, SCALE(I) = 1.0 for i = 1,...,N; = 'P': permute only; = 'S': scale only; = 'B': both permute and scale.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the input matrix A. On exit, A is overwritten by the balanced matrix. If JOB = 'N', A is not referenced. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

ILO Output parameter.

ILO is INTEGER

IHI Output parameter.

IHI is INTEGER

ILO and IHI are set to INTEGER such that on exit A(i,j) = 0 if i > j and j = 1,...,ILO-1 or I = IHI+1,...,N. If JOB = 'N' or 'S', ILO = 1 and IHI = N.

SCALE Output parameter.

SCALE is DOUBLE PRECISION

SCALE is an array, dimension (N). Details of the permutations and scaling factors applied to A. If P(j) is the index of the row and column interchanged with row and column j and D(j) is the scaling factor applied to row and column j, then SCALE(j) = P(j) for j = 1,...,ILO-1 = D(j) for j = ILO,...,IHI = P(j) for j = IHI+1,...,N. The order in which the interchanges are made is N to IHI+1, then 1 to ILO-1.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cgebal*, *dgebal* and *sgebal*. It also exists with a native C interface as *LAPACKE_zgebal*.

4.13.51 zgees

zgees computes for an N-by-N complex nonsymmetric matrix A, the eigenvalues, the Schur form T, and, optionally, the matrix of Schur vectors Z. This gives the Schur factorization A = Z*T*(Z^H).

Optionally, it also orders the eigenvalues on the diagonal of the Schur form so that selected eigenvalues are at the top left. The leading columns of Z then form an orthonormal basis for the invariant subspace corresponding to the selected eigenvalues.

A complex matrix is in Schur form if it is upper triangular.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgees(JOBVS, SORT, SELECT, N, A, LDA, SDIM, W, VS, LDVS, WORK,
                 LWORK, RWORK, BWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgees_(const char *jobvs, const char *sort, ARMPL_ZGEES_SELECT select,
            const armpl_int_t *n, armpl_doublecomplex_t *a,
            const armpl_int_t *lda, armpl_int_t *sdim,
            armpl_doublecomplex_t *w, armpl_doublecomplex_t *vs,
            const armpl_int_t *ldvs, armpl_doublecomplex_t *work,
            const armpl_int_t *lwork, double *rwork, armpl_int_t *bwork,
            armpl_int_t *info, ... );
```

Parameters

JOBVS Input parameter.

JOBVS is CHARACTER*1

= 'N': Schur vectors are not computed; = 'V': Schur vectors are computed.

SORT Input parameter.

SORT is CHARACTER*1

Specifies whether or not to order the eigenvalues on the diagonal of the Schur form. = 'N': Eigenvalues are not ordered: = 'S': Eigenvalues are ordered (see SELECT).

SELECT Input parameter.

SELECT is a LOGICAL FUNCTION of one COMPLEX*16 argument

SELECT must be declared EXTERNAL in the calling subroutine. If SORT = 'S', SELECT is used to select eigenvalues to order to the top left of the Schur form. IF SORT = 'N', SELECT is not referenced. The eigenvalue W(j) is selected if SELECT(W(j)) is true.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A. On exit, A has been overwritten by its Schur form T.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

SDIM Output parameter.

SDIM is INTEGER

If SORT = 'N', SDIM = 0. If SORT = 'S', SDIM = number of eigenvalues for which SELECT is true.

W Output parameter.

W is COMPLEX*16

W is an array, dimension (N). W contains the computed eigenvalues, in the same order that they appear on the diagonal of the output Schur form T.

VS Output parameter.

VS is COMPLEX*16

VS is an array, dimension (LDVS, N). If JOBVS = 'V', VS contains the unitary matrix Z of Schur vectors. If JOBVS = 'N', VS is not referenced.

LDVS Input parameter.

LDVS is INTEGER

The leading dimension of the array VS. LDVS >= 1; if JOBVS = 'V', LDVS >= N.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,2*N). For good performance, LWORK must generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

BWORK Output parameter.

BWORK is LOGICAL

BWORK is an array, dimension (N). Not referenced if SORT = 'N'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, and i is <= N: the QR algorithm failed to compute all the eigenvalues; elements 1:ILO-1 and i+1:N of W contain those eigenvalues which have converged; if JOBVS = 'V', VS contains the matrix which reduces A to its partially converged Schur form. = N+1: the eigenvalues could not be reordered because some eigenvalues were too close to separate (the problem is very ill-conditioned); = N+2: after reordering, roundoff changed values of some complex eigenvalues so that leading eigenvalues in the Schur form no longer satisfy SELECT = .TRUE.. This could also be caused by underflow due to scaling.

Related Information

For this routine in other precisions, please see *cgees*, *dgees* and *sgees*. It also exists with a native C interface as *LAPACKE_zgees*.

4.13.52 zgeesx

zgeesx computes for an N-by-N complex nonsymmetric matrix A, the eigenvalues, the Schur form T, and, optionally, the matrix of Schur vectors Z. This gives the Schur factorization A = Z*T*(Z^H).

Optionally, it also orders the eigenvalues on the diagonal of the Schur form so that selected eigenvalues are at the top left; computes a reciprocal condition number for the average of the selected eigenvalues (RCONDE); and computes a reciprocal condition number for the right invariant subspace corresponding to the selected eigenvalues (RCONDV). The leading columns of Z form an orthonormal basis for this invariant subspace.

For further explanation of the reciprocal condition numbers RCONDE and RCONDV, see Section 4.10 of the LAPACK Users' Guide (where these quantities are called s and sep respectively).

A complex matrix is in Schur form if it is upper triangular.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgeesx(JOBVS, SORT, SELECT, SENSE, N, A, LDA, SDIM, W, VS, LDVS,
                  RCONDE, RCONDV, WORK, LWORK, RWORK, BWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgeesx_(const char *jobvs, const char *sort, ARMPL_ZGEESX_SELECT select,
             const char *sense, const armpl_int_t *n,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_int_t *sdim, armpl_doublecomplex_t *w,
             armpl_doublecomplex_t *vs, const armpl_int_t *ldvs,
             double *rconde, double *rcondv, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, double *rwork, armpl_int_t *bwork,
             armpl_int_t *info, ... );
```

Parameters

JOBVS Input parameter.

JOBVS is CHARACTER*1

= 'N': Schur vectors are not computed; = 'V': Schur vectors are computed.

SORT Input parameter.

SORT is CHARACTER*1

Specifies whether or not to order the eigenvalues on the diagonal of the Schur form. = 'N': Eigenvalues are not ordered; = 'S': Eigenvalues are ordered (see SELECT).

SELECT Input parameter.

SELECT is a LOGICAL FUNCTION of one COMPLEX*16 argument

SELECT must be declared EXTERNAL in the calling subroutine. If SORT = 'S', SELECT is used to select eigenvalues to order to the top left of the Schur form. If SORT = 'N', SELECT is not referenced. An eigenvalue W(j) is selected if SELECT(W(j)) is true.

SENSE Input parameter.

SENSE is CHARACTER*1

Determines which reciprocal condition numbers are computed. = 'N': None are computed; = 'E': Computed for average of selected eigenvalues only; = 'V': Computed for selected right invariant subspace only; = 'B': Computed for both. If SENSE = 'E', 'V' or 'B', SORT must equal 'S'.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A. On exit, A is overwritten by its Schur form T.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

SDIM Output parameter.

SDIM is INTEGER

If SORT = 'N', SDIM = 0. If SORT = 'S', SDIM = number of eigenvalues for which SELECT is true.

W Output parameter.

W is COMPLEX*16

W is an array, dimension (N). W contains the computed eigenvalues, in the same order that they appear on the diagonal of the output Schur form T.

VS Output parameter.

VS is COMPLEX*16

VS is an array, dimension (LDVS, N). If JOBVS = 'V', VS contains the unitary matrix Z of Schur vectors. If JOBVS = 'N', VS is not referenced.

LDVS Input parameter.

LDVS is INTEGER

The leading dimension of the array VS. LDVS >= 1, and if JOBVS = 'V', LDVS >= N.

RCONDE Output parameter.

RCONDE is DOUBLE PRECISION

If SENSE = 'E' or 'B', RCONDE contains the reciprocal condition number for the average of the selected eigenvalues. Not referenced if SENSE = 'N' or 'V'.

RCONDV Output parameter.

RCONDV is DOUBLE PRECISION

If SENSE = 'V' or 'B', RCONDV contains the reciprocal condition number for the selected right invariant subspace. Not referenced if SENSE = 'N' or 'E'.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,2*N). Also, if SENSE = 'E' or 'V' or 'B', LWORK >= 2*SDIM*(N-SDIM), where SDIM is the number of selected eigenvalues computed by this routine. Note that 2*SDIM*(N-SDIM) <= N*N/2. Note also that an error is only returned if LWORK < max(1,2*N), but if SENSE = 'E' or 'V' or 'B' this may not be large enough. For good performance, LWORK must generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates upper bound on the optimal size of the array WORK, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

BWORK Output parameter.

BWORK is LOGICAL

BWORK is an array, dimension (N). Not referenced if SORT = 'N'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, and i is <= N: the QR algorithm failed to compute all the eigenvalues; elements 1:ILO-1 and i+1:N of W contain those eigenvalues which have converged; if JOBVS = 'V', VS contains the transformation which reduces A to its partially converged Schur form. = N+1: the eigenvalues could not be reordered because some eigenvalues were too close to separate (the problem is very ill-conditioned); = N+2: after reordering, roundoff changed values of some complex eigenvalues so that leading eigenvalues in the Schur form no longer satisfy SELECT=.TRUE. This could also be caused by underflow due to scaling.

Related Information

For this routine in other precisions, please see *cgeesx*, *dgeesx* and *sgeesx*. It also exists with a native C interface as *LAPACKE_zgeesx*.

4.13.53 zgeev

zgeev computes for an N-by-N complex nonsymmetric matrix A, the eigenvalues and, optionally, the left and/or right eigenvectors.

The right eigenvector v(j) of A satisfies

```
A * v(j) = lambda(j) * v(j)
```

where lambda(j) is its eigenvalue. The left eigenvector u(j) of A satisfies

```
u(j)**H * A = lambda(j) * u(j)**H
```

where $u(j)^H$ denotes the conjugate transpose of u(j).

The computed eigenvectors are normalized to have Euclidean norm equal to 1 and largest component real.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgeev(JOBVL, JOBVR, N, A, LDA, W, VL, LDVL, VR, LDVR, WORK, LWORK,
                 RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgeev_(const char *jobvl, const char *jobvr, const armpl_int_t *n,
            armpl_doublecomplex_t *a, const armpl_int_t *lda,
            armpl_doublecomplex_t *w, armpl_doublecomplex_t *vl,
```

(continues on next page)

(continued from previous page)

```
            const armpl_int_t *ldvl, armpl_doublecomplex_t *vr,
            const armpl_int_t *ldvr, armpl_doublecomplex_t *work,
            const armpl_int_t *lwork, double *rwork, armpl_int_t *info,
            ... );
```

Parameters

JOBVL Input parameter.

JOBVL is CHARACTER*1

= 'N': left eigenvectors of A are not computed; = 'V': left eigenvectors of are computed.

JOBVR Input parameter.

JOBVR is CHARACTER*1

= 'N': right eigenvectors of A are not computed; = 'V': right eigenvectors of A are computed.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A. On exit, A has been overwritten.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

W Output parameter.

W is COMPLEX*16

W is an array, dimension (N). W contains the computed eigenvalues.

VL Output parameter.

VL is COMPLEX*16

VL is an array, dimension (LDVL, N). If JOBVL = 'V', the left eigenvectors u(j) are stored one after another in the columns of VL, in the same order as their eigenvalues. If JOBVL = 'N', VL is not referenced. u(j) = VL(:,j), the j-th column of VL.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the array VL. LDVL >= 1; if JOBVL = 'V', LDVL >= N.

VR Output parameter.

VR is COMPLEX*16

VR is an array, dimension (LDVR, N). If JOBVR = 'V', the right eigenvectors v(j) are stored one after another in the columns of VR, in the same order as their eigenvalues. If JOBVR = 'N', VR is not referenced. v(j) = VR(:,j), the j-th column of VR.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the array VR. LDVR >= 1; if JOBVR = 'V', LDVR >= N.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,2*N). For good performance, LWORK must generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, the QR algorithm failed to compute all the eigenvalues, and no eigenvectors have been computed; elements and i+1:N of W contain eigenvalues which have converged.

Related Information

For this routine in other precisions, please see *cgeev*, *dgeev* and *sgeev*. It also exists with a native C interface as *LAPACKE_zgeev*.

4.13.54 zgeevx

zgeevx computes for an N-by-N complex nonsymmetric matrix A, the eigenvalues and, optionally, the left and/or right eigenvectors.

Optionally also, it computes a balancing transformation to improve the conditioning of the eigenvalues and eigenvectors (ILO, IHI, SCALE, and ABNRM), reciprocal condition numbers for the eigenvalues (RCONDE), and reciprocal condition numbers for the right eigenvectors (RCONDV).

The right eigenvector v(j) of A satisfies

```
A * v(j) = lambda(j) * v(j)
```

where lambda(j) is its eigenvalue. The left eigenvector u(j) of A satisfies

```
u(j)**H * A = lambda(j) * u(j)**H
```

where $u(j)^H$ denotes the conjugate transpose of u(j).

The computed eigenvectors are normalized to have Euclidean norm equal to 1 and largest component real.

Balancing a matrix means permuting the rows and columns to make it more nearly upper triangular, and applying a diagonal similarity transformation D * A * D**(-1), where D is a diagonal matrix, to make its rows and columns closer in norm and the condition numbers of its eigenvalues and eigenvectors smaller. The computed reciprocal condition numbers correspond to the balanced matrix. Permuting rows and columns will not change the condition numbers (in exact arithmetic) but diagonal scaling will. For further explanation of balancing, see section 4.10.2 of the LAPACK Users' Guide.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgeevx(BALANC, JOBVL, JOBVR, SENSE, N, A, LDA, W, VL, LDVL, VR,
                  LDVR, ILO, IHI, SCALE, ABNRM, RCONDE, RCONDV, WORK, LWORK,
                  RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgeevx_(const char *balanc, const char *jobvl, const char *jobvr,
             const char *sense, const armpl_int_t *n,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_doublecomplex_t *w, armpl_doublecomplex_t *vl,
             const armpl_int_t *ldvl, armpl_doublecomplex_t *vr,
             const armpl_int_t *ldvr, armpl_int_t *ilo, armpl_int_t *ihi,
             double *scale, double *abnrm, double *rconde, double *rcondv,
             armpl_doublecomplex_t *work, const armpl_int_t *lwork,
             double *rwork, armpl_int_t *info, ... );
```

Parameters

BALANC Input parameter.

BALANC is CHARACTER*1

Indicates how the input matrix should be diagonally scaled and/or permuted to improve the conditioning of its eigenvalues. = 'N': Do not diagonally scale or permute; = 'P': Perform permutations to make the matrix more nearly upper triangular. Do not diagonally scale; = 'S': Diagonally scale the matrix, ie. replace A by D*A*D**(-1), where D is a diagonal matrix chosen to make the rows and columns of A more equal in norm. Do not permute; = 'B': Both diagonally scale and permute A.

Computed reciprocal condition numbers will be for the matrix after balancing and/or permuting. Permuting does not change condition numbers (in exact arithmetic), but balancing does.

JOBVL Input parameter.

JOBVL is CHARACTER*1

= 'N': left eigenvectors of A are not computed; = 'V': left eigenvectors of A are computed. If SENSE = 'E' or 'B', JOBVL must = 'V'.

JOBVR Input parameter.

JOBVR is CHARACTER*1

= 'N': right eigenvectors of A are not computed; = 'V': right eigenvectors of A are computed. If SENSE = 'E' or 'B', JOBVR must = 'V'.

SENSE Input parameter.

SENSE is CHARACTER*1

Determines which reciprocal condition numbers are computed. = 'N': None are computed; = 'E': Computed for eigenvalues only; = 'V': Computed for right eigenvectors only; = 'B': Computed for eigenvalues and right eigenvectors.

If SENSE = 'E' or 'B', both left and right eigenvectors must also be computed (JOBVL = 'V' and JOBVR = 'V').

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A. On exit, A has been overwritten. If JOBVL = 'V' or JOBVR = 'V', A contains the Schur form of the balanced version of the matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

W Output parameter.

W is COMPLEX*16

W is an array, dimension (N). W contains the computed eigenvalues.

VL Output parameter.

VL is COMPLEX*16

VL is an array, dimension (LDVL, N). If JOBVL = 'V', the left eigenvectors u(j) are stored one after another in the columns of VL, in the same order as their eigenvalues. If JOBVL = 'N', VL is not referenced. u(j) = VL(:,j), the j-th column of VL.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the array VL. LDVL >= 1; if JOBVL = 'V', LDVL >= N.

VR Output parameter.

VR is COMPLEX*16

VR is an array, dimension (LDVR, N). If JOBVR = 'V', the right eigenvectors v(j) are stored one after another in the columns of VR, in the same order as their eigenvalues. If JOBVR = 'N', VR is not referenced. v(j) = VR(:,j), the j-th column of VR.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the array VR. LDVR >= 1; if JOBVR = 'V', LDVR >= N.

ILO Output parameter.

ILO is INTEGER

IHI Output parameter.

IHI is INTEGER

ILO and IHI are integer values determined when A was balanced. The balanced A(i,j) = 0 if I > J and J = 1,...,ILO-1 or I = IHI+1,...,N.

SCALE Output parameter.

SCALE is DOUBLE PRECISION

SCALE is an array, dimension (N). Details of the permutations and scaling factors applied when balancing A. If P(j) is the index of the row and column interchanged with row and column j, and D(j) is the scaling factor applied to row and column j, then SCALE(J) = P(J), for J = 1,...,ILO-1 = D(J), for J = ILO,...,IHI = P(J) for J = IHI+1,...,N. The order in which the interchanges are made is N to IHI+1, then 1 to ILO-1.

ABNRM Output parameter.

ABNRM is DOUBLE PRECISION

The one-norm of the balanced matrix (the maximum of the sum of absolute values of elements of any column).

RCONDE Output parameter.

RCONDE is DOUBLE PRECISION

RCONDE is an array, dimension (N). RCONDE(j) is the reciprocal condition number of the j-th eigenvalue.

RCONDV Output parameter.

RCONDV is DOUBLE PRECISION

RCONDV is an array, dimension (N). RCONDV(j) is the reciprocal condition number of the j-th right eigenvector.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SENSE = 'N' or 'E', LWORK >= max(1,2*N), and if SENSE = 'V' or 'B', LWORK >= N*N+2*N. For good performance, LWORK must generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, the QR algorithm failed to compute all the eigenvalues, and no eigenvectors or condition numbers have been computed; elements 1:ILO-1 and i+1:N of W contain eigenvalues which have converged.

Related Information

For this routine in other precisions, please see *cgeevx*, *dgeevx* and *sgeevx*. It also exists with a native C interface as *LAPACKE_zgeevx*.

4.13.55 zgehrd

zgehrd reduces a complex general matrix A to upper Hessenberg form H by an unitary similarity transformation: $Q^H * A * Q = H$.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgehrd(N, ILO, IHI, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgehrd_(const armpl_int_t *n, const armpl_int_t *ilo,
             const armpl_int_t *ihi, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_doublecomplex_t *tau,
             armpl_doublecomplex_t *work, const armpl_int_t *lwork,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

It is assumed that A is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N. ILO and IHI are normally set by a previous call to ZGEBAL; otherwise they should be set to 1 and N respectively. See Further Details. 1 <= ILO <= IHI <= N, if N > 0; ILO=1 and IHI=0, if N=0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the N-by-N general matrix to be reduced. On exit, the upper triangle and the first subdiagonal of A are overwritten with the upper Hessenberg matrix H, and the elements below the first subdiagonal, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TAU Output parameter.

TAU is COMPLEX*16

TAU is an array, dimension (N-1). The scalar factors of the elementary reflectors (see Further Details). Elements 1:ILO-1 and IHI:N-1 of TAU are set to zero.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (LWORK). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= max(1, N). For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cgehrd*, *dgehrd* and *sgehrd*. It also exists with a native C interface as *LAPACKE_zgehrd*.

4.13.56 zhsein

zhsein uses inverse iteration to find specified right and/or left eigenvectors of a complex upper Hessenberg matrix H.

The right eigenvector x and the left eigenvector y of the matrix H corresponding to an eigenvalue w are defined by:

```
H * x = w * x,     y**h * H = w * y**h
```

where y**h denotes the conjugate transpose of the vector y.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhsein(SIDE, EIGSRC, INITV, SELECT, N, H, LDH, W, VL, LDVL, VR,
                  LDVR, MM, M, WORK, RWORK, IFAILL, IFAILR, INFO)
```

C specification:

```
#include "armpl.h"

void zhsein_(const char *side, const char *eigsrc, const char *initv,
             const armpl_int_t *select, const armpl_int_t *n,
             const armpl_doublecomplex_t *h, const armpl_int_t *ldh,
             armpl_doublecomplex_t *w, armpl_doublecomplex_t *vl,
             const armpl_int_t *ldvl, armpl_doublecomplex_t *vr,
             const armpl_int_t *ldvr, const armpl_int_t *mm, armpl_int_t *m,
             armpl_doublecomplex_t *work, double *rwork, armpl_int_t *ifaill,
             armpl_int_t *ifailr, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'R': compute right eigenvectors only; = 'L': compute left eigenvectors only; = 'B': compute both right and left eigenvectors.

EIGSRC Input parameter.

EIGSRC is CHARACTER*1

Specifies the source of eigenvalues supplied in W: = 'Q': the eigenvalues were found using ZHSEQR; thus, if H has zero subdiagonal elements, and so is block-triangular, then the j-th eigenvalue can be assumed to be an eigenvalue of the block containing the j-th row/column. This property allows ZHSEIN to perform inverse iteration on just one diagonal block. = 'N': no assumptions are made on the correspondence between eigenvalues and diagonal blocks. In this case, ZHSEIN must always perform inverse iteration using the whole matrix H.

INITV Input parameter.

INITV is CHARACTER*1

= 'N': no initial vectors are supplied; = 'U': user-supplied initial vectors are stored in the arrays VL and/or VR.

SELECT Input parameter.

SELECT is LOGICAL

SELECT is an array, dimension (N). Specifies the eigenvectors to be computed. To select the eigenvector corresponding to the eigenvalue W(j), SELECT(j) must be set to .TRUE..

N Input parameter.

N is INTEGER

The order of the matrix H. N >= 0.

H Input parameter.

H is COMPLEX*16

H is an array, dimension (LDH, N). The upper Hessenberg matrix H. If a NaN is detected in H, the routine will return with INFO=-6.

LDH Input parameter.

LDH is INTEGER

The leading dimension of the array H. LDH >= max(1, N).

W Input and output parameter.

W is COMPLEX*16

W is an array, dimension (N). On entry, the eigenvalues of H. On exit, the real parts of W may have been altered since close eigenvalues are perturbed slightly in searching for independent eigenvectors.

VL Input and output parameter.

VL is COMPLEX*16

VL is an array, dimension (LDVL, MM). On entry, if INITV = 'U' and SIDE = 'L' or 'B', VL must contain starting vectors for the inverse iteration for the left eigenvectors; the starting vector for each eigenvector must be in the same column in which the eigenvector will be stored. On exit, if SIDE = 'L' or 'B', the left eigenvectors specified by SELECT will be stored consecutively in the columns of VL, in the same order as their eigenvalues. If SIDE = 'R', VL is not referenced.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the array VL. LDVL >= max(1, N) if SIDE = 'L' or 'B'; LDVL >= 1 otherwise.

VR Input and output parameter.

VR is COMPLEX*16

VR is an array, dimension (LDVR, MM). On entry, if INITV = 'U' and SIDE = 'R' or 'B', VR must contain starting vectors for the inverse iteration for the right eigenvectors; the starting vector for each eigenvector must be in the same column in which the eigenvector will be stored. On exit, if SIDE = 'R' or 'B', the right eigenvectors specified by SELECT will be stored consecutively in the columns of VR, in the same order as their eigenvalues. If SIDE = 'L', VR is not referenced.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the array VR. LDVR >= max(1, N) if SIDE = 'R' or 'B'; LDVR >= 1 otherwise.

MM Input parameter.

MM is INTEGER

The number of columns in the arrays VL and/or VR. MM >= M.

M Output parameter.

M is INTEGER

The number of columns in the arrays VL and/or VR required to store the eigenvectors (= the number of .TRUE. elements in SELECT).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

IFAILL Output parameter.

IFAILL is INTEGER array, dimension (MM)

If SIDE = 'L' or 'B', IFAILL(i) = j > 0 if the left eigenvector in the i-th column of VL (corresponding to the eigenvalue w(j)) failed to converge; IFAILL(i) = 0 if the eigenvector converged satisfactorily. If SIDE = 'R', IFAILL is not referenced.

IFAILR Output parameter.

IFAILR is INTEGER array, dimension (MM)

If SIDE = 'R' or 'B', IFAILR(i) = j > 0 if the right eigenvector in the i-th column of VR (corresponding to the eigenvalue w(j)) failed to converge; IFAILR(i) = 0 if the eigenvector converged satisfactorily. If SIDE = 'L', IFAILR is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, i is the number of eigenvectors which failed to converge; see IFAILL and IFAILR for further details.

Related Information

For this routine in other precisions, please see *chsein*, *dhsein* and *shsein*. It also exists with a native C interface as *LAPACKE_zhsein*.

4.13.57 zhseqr

```
ZHSEQR computes the eigenvalues of a Hessenberg matrix H
   and, optionally, the matrices T and Z from the Schur decomposition
   H = Z T Z**H, where T is an upper triangular matrix (the
   Schur form), and Z is the unitary matrix of Schur vectors.

   Optionally Z may be postmultiplied into an input unitary
   matrix Q so that this routine can give the Schur factorization
   of a matrix A which has been reduced to the Hessenberg form H
   by the unitary matrix Q:  A = Q*H*Q**H = (QZ)*T*(QZ)**H.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zhseqr(JOB, COMPZ, N, ILO, IHI, H, LDH, W, Z, LDZ, WORK, LWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void zhseqr_(const char *job, const char *compz, const armpl_int_t *n,
             const armpl_int_t *ilo, const armpl_int_t *ihi,
             armpl_doublecomplex_t *h, const armpl_int_t *ldh,
             armpl_doublecomplex_t *w, armpl_doublecomplex_t *z,
             const armpl_int_t *ldz, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

JOB Input parameter.

JOB is CHARACTER*1

= 'E': compute eigenvalues only; = 'S': compute eigenvalues and the Schur form T.

COMPZ Input parameter.

COMPZ is CHARACTER*1

= 'N': no Schur vectors are computed; = 'I': Z is initialized to the unit matrix and the matrix Z of Schur vectors of H is returned; = 'V': Z must contain an unitary matrix Q on entry, and the product Q* Z is returned.

N Input parameter.

N is INTEGER

The order of the matrix H. N .GE. 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

It is assumed that H is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N. ILO and IHI are normally set by a previous call to ZGEBAL, and then passed to ZGEHRD when the matrix output by

ZGEBAL is reduced to Hessenberg form. Otherwise ILO and IHI should be set to 1 and N respectively. If N.GT.0, then 1.LE.ILO.LE.IHI.LE.N. If N = 0, then ILO = 1 and IHI = 0.

H Input and output parameter.

H is COMPLEX*16

H is an array, dimension (LDH, N). On entry, the upper Hessenberg matrix H. On exit, if INFO = 0 and JOB = 'S', H contains the upper triangular matrix T from the Schur decomposition (the Schur form). If INFO = 0 and JOB = 'E', the contents of H are unspecified on exit. (The output value of H when INFO.GT.0 is given under the description of INFO below.)

Unlike earlier versions of ZHSEQR, this subroutine may explicitly H(i,j) = 0 for i.GT.j and j = 1, 2, ... ILO-1 or j = IHI+1, IHI+2, ... N.

LDH Input parameter.

LDH is INTEGER

The leading dimension of the array H. LDH .GE. max(1, N).

W Output parameter.

W is COMPLEX*16

W is an array, dimension (N). The computed eigenvalues. If JOB = 'S', the eigenvalues are stored in the same order as on the diagonal of the Schur form returned in H, with W(i) = H(i,i).

Z Input and output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, N). If COMPZ = 'N', Z is not referenced. If COMPZ = 'I', on entry Z need not be set and on exit, if INFO = 0, Z contains the unitary matrix Z of the Schur vectors of H. If COMPZ = 'V', on entry Z must contain an N-by-N matrix Q, which is assumed to be equal to the unit matrix except for the submatrix Z(ILO:IHI,ILO:IHI). On exit, if INFO = 0, Z contains Q*Z. Normally Q is the unitary matrix generated by ZUNGHR after the call to ZGEHRD which formed the Hessenberg matrix H. (The output value of Z when INFO.GT.0 is given under the description of INFO below.)

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. if COMPZ = 'I' or COMPZ = 'V', then LDZ.GE.MAX(1, N). Otherwize, LDZ.GE.1.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (LWORK). On exit, if INFO = 0, WORK(1) returns an estimate of the optimal value for LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK .GE. max(1, N) is sufficient and delivers very good and sometimes optimal performance. However, LWORK as large as 11*N may be required for optimal performance. A workspace query is recommended to determine the optimal workspace size.

If LWORK = -1, then ZHSEQR does a workspace query. In this case, ZHSEQR checks the input parameters and estimates the optimal workspace size for the given values of N, ILO and IHI. The estimate is returned in WORK(1). No error message related to LWORK is issued by XERBLA. Neither H nor Z are accessed.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit .LT. 0: if INFO = -i, the i-th argument had an illegal value .GT. 0: if INFO = i, ZHSEQR failed to compute all of the eigenvalues. Elements 1:ilo-1 and i+1:n of WR and WI contain those eigenvalues which have been successfully computed. (Failures are rare.)

If INFO .GT. 0 and JOB = 'E', then on exit, the remaining unconverged eigenvalues are the eigen- values of the upper Hessenberg matrix rows and columns ILO through INFO of the final, output value of H.

If INFO .GT. 0 and JOB = 'S', then on exit

(*) (initial value of H)*U = U*(final value of H)

where U is a unitary matrix. The final value of H is upper Hessenberg and triangular in rows and columns INFO+1 through IHI.

If INFO .GT. 0 and COMPZ = 'V', then on exit

(final value of Z) = (initial value of Z)*U

where U is the unitary matrix in (*) (regard- less of the value of JOB.)

If INFO .GT. 0 and COMPZ = 'I', then on exit (final value of Z) = U where U is the unitary matrix in (*) (regard- less of the value of JOB.)

If INFO .GT. 0 and COMPZ = 'N', then Z is not accessed.

Related Information

For this routine in other precisions, please see *chseqr*, *dhseqr* and *shseqr*. It also exists with a native C interface as *LAPACKE_zhseqr*.

4.13.58 ztrevc

ztrevc computes some or all of the right and/or left eigenvectors of a complex upper triangular matrix T. Matrices of this type are produced by the Schur factorization of a complex general matrix: A = Q*T*Q^H , as computed by ZHSEQR.

The right eigenvector x and the left eigenvector y of T corresponding to an eigenvalue w are defined by:

```
T*x = w*x,     (y**H)*T = w*(y**H)
```

where y^H denotes the conjugate transpose of the vector y. The eigenvalues are not input to this routine, but are read directly from the diagonal of T.

This routine returns the matrices X and/or Y of right and left eigenvectors of T, or the products Q*X and/or Q*Y, where Q is an input matrix. If Q is the unitary factor that reduces a matrix A to Schur form T, then Q*X and Q*Y are the matrices of right and left eigenvectors of A.

Syntax

Fortran specification:

```
use armpl_library

subroutine ztrevc(SIDE, HOWMNY, SELECT, N, T, LDT, VL, LDVL, VR, LDVR, MM, M,
                  WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ztrevc_(const char *side, const char *howmny, const armpl_int_t *select,
             const armpl_int_t *n, armpl_doublecomplex_t *t,
```

(continues on next page)

(continued from previous page)

```
                const armpl_int_t *ldt, armpl_doublecomplex_t *vl,
                const armpl_int_t *ldvl, armpl_doublecomplex_t *vr,
                const armpl_int_t *ldvr, const armpl_int_t *mm, armpl_int_t *m,
                armpl_doublecomplex_t *work, double *rwork, armpl_int_t *info,
                ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'R': compute right eigenvectors only; = 'L': compute left eigenvectors only; = 'B': compute both right and left eigenvectors.

HOWMNY Input parameter.

HOWMNY is CHARACTER*1

= 'A': compute all right and/or left eigenvectors; = 'B': compute all right and/or left eigenvectors, backtransformed using the matrices supplied in VR and/or VL; = 'S': compute selected right and/or left eigenvectors, as indicated by the logical array SELECT.

SELECT Input parameter.

SELECT is LOGICAL

SELECT is an array, dimension (N). If HOWMNY = 'S', SELECT specifies the eigenvectors to be computed. The eigenvector corresponding to the j-th eigenvalue is computed if SELECT(j) = .TRUE.. Not referenced if HOWMNY = 'A' or 'B'.

N Input parameter.

N is INTEGER

The order of the matrix T. N >= 0.

T Input and output parameter.

T is COMPLEX*16

T is an array, dimension (LDT, N). The upper triangular matrix T. T is modified, but restored on exit.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

VL Input and output parameter.

VL is COMPLEX*16

VL is an array, dimension (LDVL, MM). On entry, if SIDE = 'L' or 'B' and HOWMNY = 'B', VL must contain an N-by-N matrix Q (usually the unitary matrix Q of Schur vectors returned by ZHSEQR). On exit, if SIDE = 'L' or 'B', VL contains: if HOWMNY = 'A', the matrix Y of left eigenvectors of T; if HOWMNY = 'B', the matrix Q*Y; if HOWMNY = 'S', the left eigenvectors of T specified by SELECT, stored consecutively in the columns of VL, in the same order as their eigenvalues. Not referenced if SIDE = 'R'.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the array VL. LDVL >= 1, and if SIDE = 'L' or 'B', LDVL >= N.

VR Input and output parameter.

VR is COMPLEX*16

VR is an array, dimension (LDVR, MM). On entry, if SIDE = 'R' or 'B' and HOWMNY = 'B', VR must contain an N-by-N matrix Q (usually the unitary matrix Q of Schur vectors returned by ZHSEQR). On exit, if SIDE = 'R' or 'B', VR contains: if HOWMNY = 'A', the matrix X of right eigenvectors of T; if HOWMNY = 'B', the matrix Q*X; if HOWMNY = 'S', the right eigenvectors of T specified by SELECT, stored consecutively in the columns of VR, in the same order as their eigenvalues. Not referenced if SIDE = 'L'.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the array VR. LDVR >= 1, and if SIDE = 'R' or 'B'; LDVR >= N.

MM Input parameter.

MM is INTEGER

The number of columns in the arrays VL and/or VR. MM >= M.

M Output parameter.

M is INTEGER

The number of columns in the arrays VL and/or VR actually used to store the eigenvectors. If HOWMNY = 'A' or 'B', M is set to N. Each selected eigenvector occupies one column.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctrevc*, *dtrevc* and *strevc*. It also exists with a native C interface as *LAPACKE_ztrevc*.

4.13.59 ztrexc

ztrexc reorders the Schur factorization of a complex matrix A = Q*T*Q^H , so that the diagonal element of T with row index IFST is moved to row ILST.

The Schur form T is reordered by a unitary similarity transformation Z^H *T*Z, and optionally the matrix Q of Schur vectors is updated by postmultplying it with Z.

Syntax

Fortran specification:

```
use armpl_library

subroutine ztrexc(COMPQ, N, T, LDT, Q, LDQ, IFST, ILST, INFO)
```

C specification:

```
#include "armpl.h"

void ztrexc_(const char *compq, const armpl_int_t *n,
             armpl_doublecomplex_t *t, const armpl_int_t *ldt,
             armpl_doublecomplex_t *q, const armpl_int_t *ldq,
             const armpl_int_t *ifst, const armpl_int_t *ilst,
             armpl_int_t *info, ... );
```

Parameters

COMPQ Input parameter.

COMPQ is CHARACTER*1

= 'V': update the matrix Q of Schur vectors; = 'N': do not update Q.

N Input parameter.

N is INTEGER

The order of the matrix T. N >= 0. If N == 0 arguments ILST and IFST may be any value.

T Input and output parameter.

T is COMPLEX*16

T is an array, dimension (LDT, N). On entry, the upper triangular matrix T. On exit, the reordered upper triangular matrix.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

Q Input and output parameter.

Q is COMPLEX*16

Q is an array, dimension (LDQ, N). On entry, if COMPQ = 'V', the matrix Q of Schur vectors. On exit, if COMPQ = 'V', Q has been postmultiplied by the unitary transformation matrix Z which reorders T. If COMPQ = 'N', Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= 1, and if COMPQ = 'V', LDQ >= max(1, N).

IFST Input parameter.

IFST is INTEGER

ILST Input parameter.

ILST is INTEGER

Specify the reordering of the diagonal elements of T: The element with row index IFST is moved to row ILST by a sequence of transpositions between adjacent elements. 1 <= IFST <= N; 1 <= ILST <= N.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctrexc*, *dtrexc* and *strexc*. It also exists with a native C interface as *LAPACKE_ztrexc*.

4.13.60 ztrsen

`ztrsen` reorders the Schur factorization of a complex matrix A = Q*T*Q^H , so that a selected cluster of eigenvalues appears in the leading positions on the diagonal of the upper triangular matrix T, and the leading columns of Q form an orthonormal basis of the corresponding right invariant subspace.

Optionally the routine computes the reciprocal condition numbers of the cluster of eigenvalues and/or the invariant subspace.

Syntax

Fortran specification:

```
use armpl_library

subroutine ztrsen(JOB, COMPQ, SELECT, N, T, LDT, Q, LDQ, W, M, S, SEP, WORK,
                  LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ztrsen_(const char *job, const char *compq, const armpl_int_t *select,
             const armpl_int_t *n, armpl_doublecomplex_t *t,
             const armpl_int_t *ldt, armpl_doublecomplex_t *q,
             const armpl_int_t *ldq, armpl_doublecomplex_t *w, armpl_int_t *m,
             double *s, double *sep, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

JOB Input parameter.

JOB is CHARACTER*1

Specifies whether condition numbers are required for the cluster of eigenvalues (S) or the invariant subspace (SEP): = 'N': none; = 'E': for eigenvalues only (S); = 'V': for invariant subspace only (SEP); = 'B': for both eigenvalues and invariant subspace (S and SEP).

COMPQ Input parameter.

COMPQ is CHARACTER*1

= 'V': update the matrix Q of Schur vectors; = 'N': do not update Q.

SELECT Input parameter.

SELECT is LOGICAL

SELECT is an array, dimension (N). SELECT specifies the eigenvalues in the selected cluster. To select the j-th eigenvalue, SELECT(j) must be set to .TRUE..

N Input parameter.

N is INTEGER

The order of the matrix T. N >= 0.

T Input and output parameter.

T is COMPLEX*16

T is an array, dimension (LDT, N). On entry, the upper triangular matrix T. On exit, T is overwritten by the reordered matrix T, with the selected eigenvalues as the leading diagonal elements.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

Q Input and output parameter.

Q is COMPLEX*16

Q is an array, dimension (LDQ, N). On entry, if COMPQ = 'V', the matrix Q of Schur vectors. On exit, if COMPQ = 'V', Q has been postmultiplied by the unitary transformation matrix which reorders T; the leading M columns of Q form an orthonormal basis for the specified invariant subspace. If COMPQ = 'N', Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= 1; and if COMPQ = 'V', LDQ >= N.

W Output parameter.

W is COMPLEX*16

W is an array, dimension (N). The reordered eigenvalues of T, in the same order as they appear on the diagonal of T.

M Output parameter.

M is INTEGER

The dimension of the specified invariant subspace. 0 <= M <= N.

S Output parameter.

S is DOUBLE PRECISION

If JOB = 'E' or 'B', S is a lower bound on the reciprocal condition number for the selected cluster of eigenvalues. S cannot underestimate the true reciprocal condition number by more than a factor of sqrt(N). If M = 0 or N, S = 1. If JOB = 'N' or 'V', S is not referenced.

SEP Output parameter.

SEP is DOUBLE PRECISION

If JOB = 'V' or 'B', SEP is the estimated reciprocal condition number of the specified invariant subspace. If M = 0 or N, SEP = norm(T). If JOB = 'N' or 'E', SEP is not referenced.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If JOB = 'N', LWORK >= 1; if JOB = 'E', LWORK = max(1,M*(N-M)); if JOB = 'V' or 'B', LWORK >= max(1,2*M*(N-M)).

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctrsen*, *dtrsen* and *strsen*. It also exists with a native C interface as *LAPACKE_ztrsen*.

4.13.61 ztrsna

ztrsna estimates reciprocal condition numbers for specified eigenvalues and/or right eigenvectors of a complex upper triangular matrix T (or of any matrix Q*T*Q^H with Q unitary).

Syntax

Fortran specification:

```
use armpl_library

subroutine ztrsna(JOB, HOWMNY, SELECT, N, T, LDT, VL, LDVL, VR, LDVR, S, SEP,
                  MM, M, WORK, LDWORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ztrsna_(const char *job, const char *howmny, const armpl_int_t *select,
             const armpl_int_t *n, const armpl_doublecomplex_t *t,
             const armpl_int_t *ldt, const armpl_doublecomplex_t *vl,
             const armpl_int_t *ldvl, const armpl_doublecomplex_t *vr,
             const armpl_int_t *ldvr, double *s, double *sep,
             const armpl_int_t *mm, armpl_int_t *m,
             armpl_doublecomplex_t *work, const armpl_int_t *ldwork,
             double *rwork, armpl_int_t *info, ... );
```

Parameters

JOB Input parameter.

JOB is CHARACTER*1

Specifies whether condition numbers are required for eigenvalues (S) or eigenvectors (SEP): = 'E': for eigenvalues only (S); = 'V': for eigenvectors only (SEP); = 'B': for both eigenvalues and eigenvectors (S and SEP).

HOWMNY Input parameter.

HOWMNY is CHARACTER*1

= 'A': compute condition numbers for all eigenpairs; = 'S': compute condition numbers for selected eigenpairs specified by the array SELECT.

SELECT Input parameter.

SELECT is LOGICAL

SELECT is an array, dimension (N). If HOWMNY = 'S', SELECT specifies the eigenpairs for which condition numbers are required. To select condition numbers for the j-th eigenpair, SELECT(j) must be set to .TRUE.. If HOWMNY = 'A', SELECT is not referenced.

N Input parameter.

N is INTEGER

The order of the matrix T. N >= 0.

T Input parameter.

T is COMPLEX*16

T is an array, dimension (LDT, N). The upper triangular matrix T.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

VL Input parameter.

VL is COMPLEX*16

VL is an array, dimension (LDVL, M). If JOB = 'E' or 'B', VL must contain left eigenvectors of T (or of any Q*T*Q^H with Q unitary), corresponding to the eigenpairs specified by HOWMNY and SELECT. The eigenvectors must be stored in consecutive columns of VL, as returned by ZHSEIN or ZTREVC. If JOB = 'V', VL is not referenced.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the array VL. LDVL >= 1; and if JOB = 'E' or 'B', LDVL >= N.

VR Input parameter.

VR is COMPLEX*16

VR is an array, dimension (LDVR, M). If JOB = 'E' or 'B', VR must contain right eigenvectors of T (or of any Q*T*Q^H with Q unitary), corresponding to the eigenpairs specified by HOWMNY and SELECT. The eigenvectors must be stored in consecutive columns of VR, as returned by ZHSEIN or ZTREVC. If JOB = 'V', VR is not referenced.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the array VR. LDVR >= 1; and if JOB = 'E' or 'B', LDVR >= N.

S Output parameter.

S is DOUBLE PRECISION

S is an array, dimension (MM). If JOB = 'E' or 'B', the reciprocal condition numbers of the selected eigenvalues, stored in consecutive elements of the array. Thus S(j), SEP(j), and the j-th columns of VL and VR all correspond to the same eigenpair (but not in general the j-th eigenpair, unless all eigenpairs are selected). If JOB = 'V', S is not referenced.

SEP Output parameter.

SEP is DOUBLE PRECISION

SEP is an array, dimension (MM). If JOB = 'V' or 'B', the estimated reciprocal condition numbers of the selected eigenvectors, stored in consecutive elements of the array. If JOB = 'E', SEP is not referenced.

MM Input parameter.

MM is INTEGER

The number of elements in the arrays S (if JOB = 'E' or 'B') and/or SEP (if JOB = 'V' or 'B'). MM >= M.

M Output parameter.

M is INTEGER

The number of elements of the arrays S and/or SEP actually used to store the estimated condition numbers. If HOWMNY = 'A', M is set to N.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (LDWORK,N+6). If JOB = 'E', WORK is not referenced.

LDWORK Input parameter.

LDWORK is INTEGER

The leading dimension of the array WORK. LDWORK >= 1; and if JOB = 'V' or 'B', LDWORK >= N.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N). If JOB = 'E', RWORK is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctrsna*, *dtrsna* and *strsna*. It also exists with a native C interface as *LAPACKE_ztrsna*.

4.13.62 ztrsyl

ztrsyl solves the complex Sylvester matrix equation:

```
op(A)*X + X*op(B) = scale*C or
op(A)*X - X*op(B) = scale*C,
```

where op(A) = A or A^H , and A and B are both upper triangular. A is M-by-M and B is N-by-N; the right hand side C and the solution X are M-by-N; and scale is an output scale factor, set <= 1 to avoid overflow in X.

Syntax

Fortran specification:

```
use armpl_library

subroutine ztrsyl(TRANA, TRANB, ISGN, M, N, A, LDA, B, LDB, C, LDC, SCALE,
                  INFO)
```

C specification:

```
#include "armpl.h"

void ztrsyl_(const char *trana, const char *tranb, const armpl_int_t *isgn,
             const armpl_int_t *m, const armpl_int_t *n,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_doublecomplex_t *b, const armpl_int_t *ldb,
             armpl_doublecomplex_t *c, const armpl_int_t *ldc, double *scale,
             armpl_int_t *info, ... );
```

Parameters

TRANA Input parameter.

TRANA is CHARACTER*1

Specifies the option op(A): = 'N': op(A) = A (No transpose) = 'C': op(A) = A^H (Conjugate transpose)

TRANB Input parameter.

TRANB is CHARACTER*1

Specifies the option op(B): = 'N': op(B) = B (No transpose) = 'C': op(B) = B^H (Conjugate transpose)

ISGN Input parameter.

ISGN is INTEGER

Specifies the sign in the equation: = +1: solve op(A)*X + X*op(B) = scale*C = -1: solve op(A)*X - X*op(B) = scale* C

M Input parameter.

M is INTEGER

The order of the matrix A, and the number of rows in the matrices X and C. M >= 0.

N Input parameter.

N is INTEGER

The order of the matrix B, and the number of columns in the matrices X and C. N >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, M). The upper triangular matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). The upper triangular matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). On entry, the M-by-N right hand side matrix C. On exit, C is overwritten by the solution matrix X.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M)

SCALE Output parameter.

SCALE is DOUBLE PRECISION

The scale factor, scale, set <= 1 to avoid overflow in X.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value = 1: A and B have common or very close eigenvalues; perturbed values were used to solve the equation (but the matrices A and B are unchanged).

Related Information

For this routine in other precisions, please see *ctrsyl*, *dtrsyl* and *strsyl*. It also exists with a native C interface as *LAPACKE_ztrsyl*.

4.13.63 zunghr

zunghr generates a complex unitary matrix Q which is defined as the product of IHI-ILO elementary reflectors of order N, as returned by ZGEHRD:

Q = H(ilo) H(ilo+1) . . . H(ihi-1).

Syntax

Fortran specification:

```
use armpl_library

subroutine zunghr(N, ILO, IHI, A, LDA, TAU, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zunghr_(const armpl_int_t *n, const armpl_int_t *ilo,
             const armpl_int_t *ihi, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, const armpl_doublecomplex_t *tau,
             armpl_doublecomplex_t *work, const armpl_int_t *lwork,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix Q. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

ILO and IHI must have the same values as in the previous call of ZGEHRD. Q is equal to the unit matrix except in the submatrix Q(ilo+1:ihi,ilo+1:ihi). 1 <= ILO <= IHI <= N, if N > 0; ILO=1 and IHI=0, if N=0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the vectors which define the elementary reflectors, as returned by ZGEHRD. On exit, the N-by-N unitary matrix Q.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TAU Input parameter.

TAU is COMPLEX*16

TAU is an array, dimension (N-1). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by ZGEHRD.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= IHI-ILO. For optimum performance LWORK >= (IHI-ILO)*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cunghr*. It also exists with a native C interface as *LAPACKE_zunghr*.

4.13.64 zunmhr

zunmhr overwrites the general complex M-by-N matrix C with

```
SIDE = 'L'     SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'C': Q^H * C C * Q^H

where Q is a complex unitary matrix of order nq, with nq = m if SIDE = 'L' and nq = n if SIDE = 'R'. Q is defined as the product of IHI-ILO elementary reflectors, as returned by ZGEHRD:

Q = H(ilo) H(ilo+1) . . . H(ihi-1).

Syntax

Fortran specification:

```
use armpl_library

subroutine zunmhr(SIDE, TRANS, M, N, ILO, IHI, A, LDA, TAU, C, LDC, WORK,
                  LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zunmhr_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *ilo,
             const armpl_int_t *ihi, const armpl_doublecomplex_t *a,
             const armpl_int_t *lda, const armpl_doublecomplex_t *tau,
             armpl_doublecomplex_t *c, const armpl_int_t *ldc,
             armpl_doublecomplex_t *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left; = 'R': apply Q or Q^H from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply Q (No transpose) = 'C': apply Q^H (Conjugate transpose)

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

ILO and IHI must have the same values as in the previous call of ZGEHRD. Q is equal to the unit matrix except in the submatrix Q(ilo+1:ihi,ilo+1:ihi). If SIDE = 'L', then 1 <= ILO <= IHI <= M, if M > 0, and ILO = 1 and IHI = 0, if M = 0; if SIDE = 'R', then 1 <= ILO <= IHI <= N, if N > 0, and ILO = 1 and IHI = 0, if N = 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension. (LDA, M) if SIDE = 'L' (LDA, N) if SIDE = 'R' The vectors which define the elementary reflectors, as returned by ZGEHRD.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M) if SIDE = 'L'; LDA >= max(1, N) if SIDE = 'R'.

TAU Input parameter.

TAU is COMPLEX*16

TAU is an array, dimension. (M-1) if SIDE = 'L' (N-1) if SIDE = 'R' TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by ZGEHRD.

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^H * C or C*Q^H or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, N); if SIDE = 'R', LWORK >= max(1, M). For optimum performance LWORK >= N*NB if SIDE = 'L', and LWORK >= M*NB if SIDE = 'R', where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cunmhr*. It also exists with a native C interface as *LAPACKE_zunmhr*.

4.14 LAPACK generalized non symmetric eigenvalues routines

4.14.1 cggbak

cggbak forms the right or left eigenvectors of a complex generalized eigenvalue problem A*x = lambda*B*x, by backward transformation on the computed eigenvectors of the balanced pair of matrices output by CGGBAL.

Syntax

Fortran specification:

```
use armpl_library

subroutine cggbak(JOB, SIDE, N, ILO, IHI, LSCALE, RSCALE, M, V, LDV, INFO)
```

C specification:

```
#include "armpl.h"

void cggbak_(const char *job, const char *side, const armpl_int_t *n,
             const armpl_int_t *ilo, const armpl_int_t *ihi,
             const float *lscale, const float *rscale, const armpl_int_t *m,
             armpl_singlecomplex_t *v, const armpl_int_t *ldv,
             armpl_int_t *info, ... );
```

Parameters

JOB Input parameter.

JOB is CHARACTER*1

Specifies the type of backward transformation required: = 'N': do nothing, return immediately; = 'P': do backward transformation for permutation only; = 'S': do backward transformation for scaling only; = 'B': do backward transformations for both permutation and scaling. JOB must be the same as the argument JOB supplied to CGGBAL.

SIDE Input parameter.

SIDE is CHARACTER*1

= 'R': V contains right eigenvectors; = 'L': V contains left eigenvectors.

N Input parameter.

N is INTEGER

The number of rows of the matrix V. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

The integers ILO and IHI determined by CGGBAL. 1 <= ILO <= IHI <= N, if N > 0; ILO=1 and IHI=0, if N=0.

LSCALE Input parameter.

LSCALE is REAL

LSCALE is an array, dimension (N). Details of the permutations and/or scaling factors applied to the left side of A and B, as returned by CGGBAL.

RSCALE Input parameter.

RSCALE is REAL

RSCALE is an array, dimension (N). Details of the permutations and/or scaling factors applied to the right side of A and B, as returned by CGGBAL.

M Input parameter.

M is INTEGER

The number of columns of the matrix V. M >= 0.

V Input and output parameter.

V is COMPLEX

V is an array, dimension (LDV, M). On entry, the matrix of right or left eigenvectors to be transformed, as returned by CTGEVC. On exit, V is overwritten by the transformed eigenvectors.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the matrix V. LDV >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *dggbak*, *sggbak* and *zggbak*. It also exists with a native C interface as *LAPACKE_cggbak*.

4.14.2 cggbal

cggbal balances a pair of general complex matrices (A,B). This involves, first, permuting A and B by similarity transformations to isolate eigenvalues in the first 1 to ILO$-$1 and last IHI+1 to N elements on the diagonal; and second, applying a diagonal similarity transformation to rows and columns ILO to IHI to make the rows and columns as close in norm as possible. Both steps are optional.

Balancing may reduce the 1-norm of the matrices, and improve the accuracy of the computed eigenvalues and/or eigenvectors in the generalized eigenvalue problem A*x = lambda*B*x.

Syntax

Fortran specification:

```
use armpl_library

subroutine cggbal(JOB, N, A, LDA, B, LDB, ILO, IHI, LSCALE, RSCALE, WORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void cggbal_(const char *job, const armpl_int_t *n, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, armpl_int_t *ilo, armpl_int_t *ihi,
             float *lscale, float *rscale, float *work, armpl_int_t *info,
             ... );
```

Parameters

JOB Input parameter.

JOB is CHARACTER*1

Specifies the operations to be performed on A and B: = 'N': none: simply set ILO = 1, IHI = N, LSCALE(I) = 1.0 and RSCALE(I) = 1.0 for i=1,...,N; = 'P': permute only; = 'S': scale only; = 'B': both permute and scale.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the input matrix A. On exit, A is overwritten by the balanced matrix. If JOB = 'N', A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, N). On entry, the input matrix B. On exit, B is overwritten by the balanced matrix. If JOB = 'N', B is not referenced.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

ILO Output parameter.

ILO is INTEGER

IHI Output parameter.

IHI is INTEGER

ILO and IHI are set to integers such that on exit A(i,j) = 0 and B(i,j) = 0 if i > j and j = 1,. . . ,ILO-1 or i = IHI+1,. . . ,N. If JOB = 'N' or 'S', ILO = 1 and IHI = N.

LSCALE Output parameter.

LSCALE is REAL

LSCALE is an array, dimension (N). Details of the permutations and scaling factors applied to the left side of A and B. If P(j) is the index of the row interchanged with row j, and D(j) is the scaling factor applied to row j, then LSCALE(j) = P(j) for J = 1,. . . ,ILO-1 = D(j) for J = ILO,. . . ,IHI = P(j) for J = IHI+1,. . . ,N. The order in which the interchanges are made is N to IHI+1, then 1 to ILO-1.

RSCALE Output parameter.

RSCALE is REAL

RSCALE is an array, dimension (N). Details of the permutations and scaling factors applied to the right side of A and B. If P(j) is the index of the column interchanged with column j, and D(j) is the scaling factor applied to column j, then RSCALE(j) = P(j) for J = 1,. . . ,ILO-1 = D(j) for J = ILO,. . . ,IHI = P(j) for J = IHI+1,. . . ,N. The order in which the interchanges are made is N to IHI+1, then 1 to ILO-1.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (lwork). lwork must be at least max(1,6*N) when JOB = 'S' or 'B', and at least 1 when JOB = 'N' or 'P'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *dggbal*, *sggbal* and *zggbal*. It also exists with a native C interface as *LAPACKE_cggbal*.

4.14.3 cgges

`cgges` computes for a pair of N-by-N complex nonsymmetric matrices (A,B), the generalized eigenvalues, the generalized complex Schur form (S, T), and optionally left and/or right Schur vectors (VSL and VSR). This gives the generalized Schur factorization

```
(A,B) = ( (VSL)*S*(VSR)**H, (VSL)*T*(VSR)**H )
```

where $(VSR)^H$ is the conjugate-transpose of VSR.

Optionally, it also orders the eigenvalues so that a selected cluster of eigenvalues appears in the leading diagonal blocks of the upper triangular matrix S and the upper triangular matrix T. The leading columns of VSL and VSR then form an unitary basis for the corresponding left and right eigenspaces (deflating subspaces).

(If only the generalized eigenvalues are needed, use the driver CGGEV instead, which is faster.)

A generalized eigenvalue for a pair of matrices (A,B) is a scalar w or a ratio alpha/beta = w, such that A - w*B is singular. It is usually represented as the pair (alpha,beta), as there is a reasonable interpretation for beta=0, and even for both being zero.

A pair of matrices (S,T) is in generalized complex Schur form if S and T are upper triangular and, in addition, the diagonal elements of T are non-negative real numbers.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgges(JOBVSL, JOBVSR, SORT, SELCTG, N, A, LDA, B, LDB, SDIM, ALPHA,
                 BETA, VSL, LDVSL, VSR, LDVSR, WORK, LWORK, RWORK, BWORK,
                 INFO)
```

C specification:

```
#include "armpl.h"

void cgges_(const char *jobvsl, const char *jobvsr, const char *sort,
            ARMPL_CGGES_SELCTG selctg, const armpl_int_t *n,
            armpl_singlecomplex_t *a, const armpl_int_t *lda,
            armpl_singlecomplex_t *b, const armpl_int_t *ldb,
            armpl_int_t *sdim, armpl_singlecomplex_t *alpha,
            armpl_singlecomplex_t *beta, armpl_singlecomplex_t *vsl,
            const armpl_int_t *ldvsl, armpl_singlecomplex_t *vsr,
            const armpl_int_t *ldvsr, armpl_singlecomplex_t *work,
            const armpl_int_t *lwork, float *rwork, armpl_int_t *bwork,
            armpl_int_t *info, ... );
```

Parameters

JOBVSL Input parameter.

JOBVSL is CHARACTER*1

= 'N': do not compute the left Schur vectors; = 'V': compute the left Schur vectors.

JOBVSR Input parameter.

JOBVSR is CHARACTER*1

= 'N': do not compute the right Schur vectors; = 'V': compute the right Schur vectors.

SORT Input parameter.

SORT is CHARACTER*1

Specifies whether or not to order the eigenvalues on the diagonal of the generalized Schur form. = 'N': Eigenvalues are not ordered; = 'S': Eigenvalues are ordered (see SELCTG).

SELCTG Input parameter.

SELCTG is a LOGICAL FUNCTION of two COMPLEX arguments

SELCTG must be declared EXTERNAL in the calling subroutine. If SORT = 'N', SELCTG is not referenced. If SORT = 'S', SELCTG is used to select eigenvalues to sort to the top left of the Schur form. An eigenvalue ALPHA(j)/BETA(j) is selected if SELCTG(ALPHA(j),BETA(j)) is true.

Note that a selected complex eigenvalue may no longer satisfy SELCTG(ALPHA(j),BETA(j)) = .TRUE. after ordering, since ordering may change the value of complex eigenvalues (especially if the eigenvalue is ill-conditioned), in this case INFO is set to N+2 (See INFO below).

N Input parameter.

N is INTEGER

The order of the matrices A, B, VSL, and VSR. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the first of the pair of matrices. On exit, A has been overwritten by its generalized Schur form S.

LDA Input parameter.

LDA is INTEGER

The leading dimension of A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, N). On entry, the second of the pair of matrices. On exit, B has been overwritten by its generalized Schur form T.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B. LDB >= max(1, N).

SDIM Output parameter.

SDIM is INTEGER

If SORT = 'N', SDIM = 0. If SORT = 'S', SDIM = number of eigenvalues (after sorting) for which SELCTG is true.

ALPHA Output parameter.

ALPHA is COMPLEX

ALPHA is an array, dimension (N) .

BETA Output parameter.

BETA is COMPLEX

BETA is an array, dimension (N). On exit, ALPHA(j)/BETA(j), j=1,..., N, will be the generalized eigenvalues. ALPHA(j), j=1,...,N and BETA(j), j=1,...,N are the diagonals of the complex Schur form (A, B) output by CGGES. The BETA(j) will be non-negative real.

Note: the quotients ALPHA(j)/BETA(j) may easily over- or underflow, and BETA(j) may even be zero. Thus, the user should avoid naively computing the ratio alpha/beta. However, ALPHA will be always less than and usually comparable with norm(A) in magnitude, and BETA always less than and usually comparable with norm(B).

VSL Output parameter.

VSL is COMPLEX

VSL is an array, dimension (LDVSL, N). If JOBVSL = 'V', VSL will contain the left Schur vectors. Not referenced if JOBVSL = 'N'.

LDVSL Input parameter.

LDVSL is INTEGER

The leading dimension of the matrix VSL. LDVSL >= 1, and if JOBVSL = 'V', LDVSL >= N.

VSR Output parameter.

VSR is COMPLEX

VSR is an array, dimension (LDVSR, N). If JOBVSR = 'V', VSR will contain the right Schur vectors. Not referenced if JOBVSR = 'N'.

LDVSR Input parameter.

LDVSR is INTEGER

The leading dimension of the matrix VSR. LDVSR >= 1, and if JOBVSR = 'V', LDVSR >= N.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,2*N). For good performance, LWORK must generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (8*N) .

BWORK Output parameter.

BWORK is LOGICAL

BWORK is an array, dimension (N). Not referenced if SORT = 'N'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. =1,...,N: The QZ iteration failed. (A, B) are not in Schur form, but ALPHA(j) and BETA(j) should be correct for j=INFO+1,...,N. > N: =N+1: other than QZ iteration failed in CHGEQZ =N+2: after reordering, roundoff changed values

of some complex eigenvalues so that leading eigenvalues in the Generalized Schur form no longer satisfy SELCTG=.TRUE. This could also be caused due to scaling. =N+3: reordering failed in CTGSEN.

Related Information

For this routine in other precisions, please see *dgges*, *sgges* and *zgges*. It also exists with a native C interface as *LAPACKE_cgges*.

4.14.4 cgges3

`cgges3` computes for a pair of N-by-N complex nonsymmetric matrices (A,B), the generalized eigenvalues, the generalized complex Schur form (S, T), and optionally left and/or right Schur vectors (VSL and VSR). This gives the generalized Schur factorization

```
(A,B) = ( (VSL)*S*(VSR)**H, (VSL)*T*(VSR)**H )
```

where $(VSR)^H$ is the conjugate-transpose of VSR.

Optionally, it also orders the eigenvalues so that a selected cluster of eigenvalues appears in the leading diagonal blocks of the upper triangular matrix S and the upper triangular matrix T. The leading columns of VSL and VSR then form an unitary basis for the corresponding left and right eigenspaces (deflating subspaces).

(If only the generalized eigenvalues are needed, use the driver CGGEV instead, which is faster.)

A generalized eigenvalue for a pair of matrices (A,B) is a scalar w or a ratio alpha/beta = w, such that A - w*B is singular. It is usually represented as the pair (alpha,beta), as there is a reasonable interpretation for beta=0, and even for both being zero.

A pair of matrices (S,T) is in generalized complex Schur form if S and T are upper triangular and, in addition, the diagonal elements of T are non-negative real numbers.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgges3(JOBVSL, JOBVSR, SORT, SELCTG, N, A, LDA, B, LDB, SDIM,
                  ALPHA, BETA, VSL, LDVSL, VSR, LDVSR, WORK, LWORK, RWORK,
                  BWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgges3_(const char *jobvsl, const char *jobvsr, const char *sort,
             ARMPL_CGGES3_SELCTG selctg, const armpl_int_t *n,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             armpl_singlecomplex_t *b, const armpl_int_t *ldb,
             armpl_int_t *sdim, armpl_singlecomplex_t *alpha,
             armpl_singlecomplex_t *beta, armpl_singlecomplex_t *vsl,
             const armpl_int_t *ldvsl, armpl_singlecomplex_t *vsr,
             const armpl_int_t *ldvsr, armpl_singlecomplex_t *work,
             const armpl_int_t *lwork, float *rwork, armpl_int_t *bwork,
             armpl_int_t *info, ... );
```

Parameters

JOBVSL Input parameter.

JOBVSL is CHARACTER*1

= 'N': do not compute the left Schur vectors; = 'V': compute the left Schur vectors.

JOBVSR Input parameter.

JOBVSR is CHARACTER*1

= 'N': do not compute the right Schur vectors; = 'V': compute the right Schur vectors.

SORT Input parameter.

SORT is CHARACTER*1

Specifies whether or not to order the eigenvalues on the diagonal of the generalized Schur form. = 'N': Eigenvalues are not ordered; = 'S': Eigenvalues are ordered (see SELCTG).

SELCTG Input parameter.

SELCTG is a LOGICAL FUNCTION of two COMPLEX arguments

SELCTG must be declared EXTERNAL in the calling subroutine. If SORT = 'N', SELCTG is not referenced. If SORT = 'S', SELCTG is used to select eigenvalues to sort to the top left of the Schur form. An eigenvalue ALPHA(j)/BETA(j) is selected if SELCTG(ALPHA(j),BETA(j)) is true.

Note that a selected complex eigenvalue may no longer satisfy SELCTG(ALPHA(j),BETA(j)) = .TRUE. after ordering, since ordering may change the value of complex eigenvalues (especially if the eigenvalue is ill-conditioned), in this case INFO is set to N+2 (See INFO below).

N Input parameter.

N is INTEGER

The order of the matrices A, B, VSL, and VSR. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the first of the pair of matrices. On exit, A has been overwritten by its generalized Schur form S.

LDA Input parameter.

LDA is INTEGER

The leading dimension of A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, N). On entry, the second of the pair of matrices. On exit, B has been overwritten by its generalized Schur form T.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B. LDB >= max(1, N).

SDIM Output parameter.

SDIM is INTEGER

If SORT = 'N', SDIM = 0. If SORT = 'S', SDIM = number of eigenvalues (after sorting) for which SELCTG is true.

ALPHA Output parameter.

ALPHA is COMPLEX

ALPHA is an array, dimension (N) .

BETA Output parameter.

BETA is COMPLEX

BETA is an array, dimension (N). On exit, ALPHA(j)/BETA(j), j=1,..., N, will be the generalized eigenvalues. ALPHA(j), j=1,...,N and BETA(j), j=1,...,N are the diagonals of the complex Schur form (A, B) output by CGGES3. The BETA(j) will be non-negative real.

Note: the quotients ALPHA(j)/BETA(j) may easily over- or underflow, and BETA(j) may even be zero. Thus, the user should avoid naively computing the ratio alpha/beta. However, ALPHA will be always less than and usually comparable with norm(A) in magnitude, and BETA always less than and usually comparable with norm(B).

VSL Output parameter.

VSL is COMPLEX

VSL is an array, dimension (LDVSL, N). If JOBVSL = 'V', VSL will contain the left Schur vectors. Not referenced if JOBVSL = 'N'.

LDVSL Input parameter.

LDVSL is INTEGER

The leading dimension of the matrix VSL. LDVSL >= 1, and if JOBVSL = 'V', LDVSL >= N.

VSR Output parameter.

VSR is COMPLEX

VSR is an array, dimension (LDVSR, N). If JOBVSR = 'V', VSR will contain the right Schur vectors. Not referenced if JOBVSR = 'N'.

LDVSR Input parameter.

LDVSR is INTEGER

The leading dimension of the matrix VSR. LDVSR >= 1, and if JOBVSR = 'V', LDVSR >= N.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (8*N) .

BWORK Output parameter.

BWORK is LOGICAL

BWORK is an array, dimension (N). Not referenced if SORT = 'N'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. =1,...,N: The QZ iteration failed. (A, B) are not in Schur form, but ALPHA(j) and BETA(j) should be correct for j=INFO+1,...,N. > N: =N+1: other than QZ iteration failed in CHGEQZ =N+2: after reordering, roundoff changed values of some complex eigenvalues so that leading eigenvalues in the Generalized Schur form no longer satisfy SELCTG=.TRUE. This could also be caused due to scaling. =N+3: reordering failed in CTGSEN.

Related Information

For this routine in other precisions, please see *dgges3*, *sgges3* and *zgges3*. It also exists with a native C interface as *LAPACKE_cgges3*.

4.14.5 cggesx

cggesx computes for a pair of N-by-N complex nonsymmetric matrices (A,B), the generalized eigenvalues, the complex Schur form (S,T), and, optionally, the left and/or right matrices of Schur vectors (VSL and VSR). This gives the generalized Schur factorization

```
(A,B) = ( (VSL) S (VSR)**H, (VSL) T (VSR)**H )
```

where $(VSR)^H$ is the conjugate-transpose of VSR.

Optionally, it also orders the eigenvalues so that a selected cluster of eigenvalues appears in the leading diagonal blocks of the upper triangular matrix S and the upper triangular matrix T; computes a reciprocal condition number for the average of the selected eigenvalues (RCONDE); and computes a reciprocal condition number for the right and left deflating subspaces corresponding to the selected eigenvalues (RCONDV). The leading columns of VSL and VSR then form an orthonormal basis for the corresponding left and right eigenspaces (deflating subspaces).

A generalized eigenvalue for a pair of matrices (A,B) is a scalar w or a ratio alpha/beta = w, such that A - w*B is singular. It is usually represented as the pair (alpha,beta), as there is a reasonable interpretation for beta=0 or for both being zero.

A pair of matrices (S,T) is in generalized complex Schur form if T is upper triangular with non-negative diagonal and S is upper triangular.

Syntax

Fortran specification:

```
use armpl_library

subroutine cggesx(JOBVSL, JOBVSR, SORT, SELCTG, SENSE, N, A, LDA, B, LDB,
                  SDIM, ALPHA, BETA, VSL, LDVSL, VSR, LDVSR, RCONDE, RCONDV,
                  WORK, LWORK, RWORK, IWORK, LIWORK, BWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cggesx_(const char *jobvsl, const char *jobvsr, const char *sort,
             ARMPL_CGGESX_SELCTG selctg, const char *sense,
             const armpl_int_t *n, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, armpl_int_t *sdim,
             armpl_singlecomplex_t *alpha, armpl_singlecomplex_t *beta,
             armpl_singlecomplex_t *vsl, const armpl_int_t *ldvsl,
```

(continues on next page)

(continued from previous page)

```
                armpl_singlecomplex_t *vsr, const armpl_int_t *ldvsr,
                float *rconde, float *rcondv, armpl_singlecomplex_t *work,
                const armpl_int_t *lwork, float *rwork, armpl_int_t *iwork,
                const armpl_int_t *liwork, armpl_int_t *bwork, armpl_int_t *info,
                ... );
```

Parameters

JOBVSL Input parameter.

JOBVSL is CHARACTER*1

= 'N': do not compute the left Schur vectors; = 'V': compute the left Schur vectors.

JOBVSR Input parameter.

JOBVSR is CHARACTER*1

= 'N': do not compute the right Schur vectors; = 'V': compute the right Schur vectors.

SORT Input parameter.

SORT is CHARACTER*1

Specifies whether or not to order the eigenvalues on the diagonal of the generalized Schur form. = 'N': Eigenvalues are not ordered; = 'S': Eigenvalues are ordered (see SELCTG).

SELCTG Input parameter.

SELCTG is a LOGICAL FUNCTION of two COMPLEX arguments

SELCTG must be declared EXTERNAL in the calling subroutine. If SORT = 'N', SELCTG is not referenced. If SORT = 'S', SELCTG is used to select eigenvalues to sort to the top left of the Schur form. Note that a selected complex eigenvalue may no longer satisfy SELCTG(ALPHA(j),BETA(j)) = .TRUE. after ordering, since ordering may change the value of complex eigenvalues (especially if the eigenvalue is ill-conditioned), in this case INFO is set to N+3 see INFO below).

SENSE Input parameter.

SENSE is CHARACTER*1

Determines which reciprocal condition numbers are computed. = 'N' : None are computed; = 'E' : Computed for average of selected eigenvalues only; = 'V' : Computed for selected deflating subspaces only; = 'B' : Computed for both. If SENSE = 'E', 'V', or 'B', SORT must equal 'S'.

N Input parameter.

N is INTEGER

The order of the matrices A, B, VSL, and VSR. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the first of the pair of matrices. On exit, A has been overwritten by its generalized Schur form S.

LDA Input parameter.

LDA is INTEGER

The leading dimension of A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, N). On entry, the second of the pair of matrices. On exit, B has been overwritten by its generalized Schur form T.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B. LDB >= max(1, N).

SDIM Output parameter.

SDIM is INTEGER

If SORT = 'N', SDIM = 0. If SORT = 'S', SDIM = number of eigenvalues (after sorting) for which SELCTG is true.

ALPHA Output parameter.

ALPHA is COMPLEX

ALPHA is an array, dimension (N) .

BETA Output parameter.

BETA is COMPLEX

BETA is an array, dimension (N). On exit, ALPHA(j)/BETA(j), j=1,..., N, will be the generalized eigenvalues. ALPHA(j) and BETA(j),j=1,...,N are the diagonals of the complex Schur form (S,T). BETA(j) will be non-negative real.

Note: the quotients ALPHA(j)/BETA(j) may easily over- or underflow, and BETA(j) may even be zero. Thus, the user should avoid naively computing the ratio alpha/beta. However, ALPHA will be always less than and usually comparable with norm(A) in magnitude, and BETA always less than and usually comparable with norm(B).

VSL Output parameter.

VSL is COMPLEX

VSL is an array, dimension (LDVSL, N). If JOBVSL = 'V', VSL will contain the left Schur vectors. Not referenced if JOBVSL = 'N'.

LDVSL Input parameter.

LDVSL is INTEGER

The leading dimension of the matrix VSL. LDVSL >=1, and if JOBVSL = 'V', LDVSL >= N.

VSR Output parameter.

VSR is COMPLEX

VSR is an array, dimension (LDVSR, N). If JOBVSR = 'V', VSR will contain the right Schur vectors. Not referenced if JOBVSR = 'N'.

LDVSR Input parameter.

LDVSR is INTEGER

The leading dimension of the matrix VSR. LDVSR >= 1, and if JOBVSR = 'V', LDVSR >= N.

RCONDE Output parameter.

RCONDE is REAL

RCONDE is an array, dimension (2). If SENSE = 'E' or 'B', RCONDE(1) and RCONDE(2) contain the reciprocal condition numbers for the average of the selected eigenvalues. Not referenced if SENSE = 'N' or 'V'.

RCONDV Output parameter.

RCONDV is REAL

RCONDV is an array, dimension (2). If SENSE = 'V' or 'B', RCONDV(1) and RCONDV(2) contain the reciprocal condition number for the selected deflating subspaces. Not referenced if SENSE = 'N' or 'E'.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If N = 0, LWORK >= 1, else if SENSE = 'E', 'V', or 'B', LWORK >= MAX(1,2*N,2*SDIM*(N-SDIM)), else LWORK >= MAX(1,2*N). Note that 2*SDIM*(N-SDIM) <= N*N/2. Note also that an error is only returned if LWORK < MAX(1,2*N), but if SENSE = 'E' or 'V' or 'B' this may not be large enough.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the bound on the optimal size of the WORK array and the minimum size of the IWORK array, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (8* N). Real workspace.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the minimum LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array WORK. If SENSE = 'N' or N = 0, LIWORK >= 1, otherwise LIWORK >= N+2.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the bound on the optimal size of the WORK array and the minimum size of the IWORK array, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

BWORK Output parameter.

BWORK is LOGICAL

BWORK is an array, dimension (N). Not referenced if SORT = 'N'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. = 1,...,N: The QZ iteration failed. (A, B) are not in Schur form, but ALPHA(j) and BETA(j) should be correct for j=INFO+1,...,N. > N: =N+1: other than QZ iteration failed in CHGEQZ =N+2: after reordering, roundoff changed values of some complex eigenvalues so that leading eigenvalues in the Generalized Schur form no longer satisfy SELCTG=.TRUE. This could also be caused due to scaling. =N+3: reordering failed in CTGSEN.

Related Information

For this routine in other precisions, please see *dggesx*, *sggesx* and *zggesx*. It also exists with a native C interface as *LAPACKE_cggesx*.

4.14.6 cggev

`cggev` computes for a pair of N-by-N complex nonsymmetric matrices (A,B), the generalized eigenvalues, and optionally, the left and/or right generalized eigenvectors.

A generalized eigenvalue for a pair of matrices (A,B) is a scalar lambda or a ratio alpha/beta = lambda, such that A - lambda*B is singular. It is usually represented as the pair (alpha,beta), as there is a reasonable interpretation for beta=0, and even for both being zero.

The right generalized eigenvector v(j) corresponding to the generalized eigenvalue lambda(j) of (A,B) satisfies

```
A * v(j) = lambda(j) * B * v(j).
```

The left generalized eigenvector u(j) corresponding to the generalized eigenvalues lambda(j) of (A,B) satisfies

```
u(j)**H * A = lambda(j) * u(j)**H * B
```

where $u(j)^H$ is the conjugate-transpose of u(j).

Syntax

Fortran specification:

```
use armpl_library

subroutine cggev(JOBVL, JOBVR, N, A, LDA, B, LDB, ALPHA, BETA, VL, LDVL, VR,
                 LDVR, WORK, LWORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cggev_(const char *jobvl, const char *jobvr, const armpl_int_t *n,
            armpl_singlecomplex_t *a, const armpl_int_t *lda,
            armpl_singlecomplex_t *b, const armpl_int_t *ldb,
            armpl_singlecomplex_t *alpha, armpl_singlecomplex_t *beta,
            armpl_singlecomplex_t *vl, const armpl_int_t *ldvl,
            armpl_singlecomplex_t *vr, const armpl_int_t *ldvr,
            armpl_singlecomplex_t *work, const armpl_int_t *lwork,
            float *rwork, armpl_int_t *info, ... );
```

Parameters

JOBVL Input parameter.

JOBVL is CHARACTER*1

= 'N': do not compute the left generalized eigenvectors; = 'V': compute the left generalized eigenvectors.

JOBVR Input parameter.

JOBVR is CHARACTER*1

= 'N': do not compute the right generalized eigenvectors; = 'V': compute the right generalized eigenvectors.

N Input parameter.

N is INTEGER

The order of the matrices A, B, VL, and VR. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the matrix A in the pair (A, B). On exit, A has been overwritten.

LDA Input parameter.

LDA is INTEGER

The leading dimension of A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, N). On entry, the matrix B in the pair (A, B). On exit, B has been overwritten.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B. LDB >= max(1, N).

ALPHA Output parameter.

ALPHA is COMPLEX

ALPHA is an array, dimension (N) .

BETA Output parameter.

BETA is COMPLEX

BETA is an array, dimension (N). On exit, ALPHA(j)/BETA(j), j=1,. . . , N, will be the generalized eigenvalues.

Note: the quotients ALPHA(j)/BETA(j) may easily over- or underflow, and BETA(j) may even be zero. Thus, the user should avoid naively computing the ratio alpha/beta. However, ALPHA will be always less than and usually comparable with norm(A) in magnitude, and BETA always less than and usually comparable with norm(B).

VL Output parameter.

VL is COMPLEX

VL is an array, dimension (LDVL, N). If JOBVL = 'V', the left generalized eigenvectors u(j) are stored one after another in the columns of VL, in the same order as their eigenvalues. Each eigenvector is scaled so the largest component has abs(real part) + abs(imag. part) = 1. Not referenced if JOBVL = 'N'.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the matrix VL. LDVL >= 1, and if JOBVL = 'V', LDVL >= N.

VR Output parameter.

VR is COMPLEX

VR is an array, dimension (LDVR, N). If JOBVR = 'V', the right generalized eigenvectors v(j) are stored one after another in the columns of VR, in the same order as their eigenvalues. Each eigenvector is scaled so the largest component has abs(real part) + abs(imag. part) = 1. Not referenced if JOBVR = 'N'.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the matrix VR. LDVR >= 1, and if JOBVR = 'V', LDVR >= N.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,2*N). For good performance, LWORK must generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (8*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. =1,. . . ,N: The QZ iteration failed. No eigenvectors have been calculated, but ALPHA(j) and BETA(j) should be correct for j=INFO+1,. . . ,N. > N: =N+1: other then QZ iteration failed in SHGEQZ, =N+2: error return from STGEVC.

Related Information

For this routine in other precisions, please see *dggev*, *sggev* and *zggev*. It also exists with a native C interface as *LAPACKE_cggev*.

4.14.7 cggev3

cggev3 computes for a pair of N-by-N complex nonsymmetric matrices (A,B), the generalized eigenvalues, and optionally, the left and/or right generalized eigenvectors.

A generalized eigenvalue for a pair of matrices (A,B) is a scalar lambda or a ratio alpha/beta = lambda, such that A - lambda*B is singular. It is usually represented as the pair (alpha,beta), as there is a reasonable interpretation for beta=0, and even for both being zero.

The right generalized eigenvector v(j) corresponding to the generalized eigenvalue lambda(j) of (A,B) satisfies

```
A * v(j) = lambda(j) * B * v(j).
```

The left generalized eigenvector u(j) corresponding to the generalized eigenvalues lambda(j) of (A,B) satisfies

```
u(j)**H * A = lambda(j) * u(j)**H * B
```

where $u(j)^H$ is the conjugate-transpose of u(j).

Syntax

Fortran specification:

```
use armpl_library

subroutine cggev3(JOBVL, JOBVR, N, A, LDA, B, LDB, ALPHA, BETA, VL, LDVL, VR,
                  LDVR, WORK, LWORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cggev3_(const char *jobvl, const char *jobvr, const armpl_int_t *n,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             armpl_singlecomplex_t *b, const armpl_int_t *ldb,
             armpl_singlecomplex_t *alpha, armpl_singlecomplex_t *beta,
             armpl_singlecomplex_t *vl, const armpl_int_t *ldvl,
             armpl_singlecomplex_t *vr, const armpl_int_t *ldvr,
             armpl_singlecomplex_t *work, const armpl_int_t *lwork,
             float *rwork, armpl_int_t *info, ... );
```

Parameters

JOBVL Input parameter.

JOBVL is CHARACTER*1

= 'N': do not compute the left generalized eigenvectors; = 'V': compute the left generalized eigenvectors.

JOBVR Input parameter.

JOBVR is CHARACTER*1

= 'N': do not compute the right generalized eigenvectors; = 'V': compute the right generalized eigenvectors.

N Input parameter.

N is INTEGER

The order of the matrices A, B, VL, and VR. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the matrix A in the pair (A, B). On exit, A has been overwritten.

LDA Input parameter.

LDA is INTEGER

The leading dimension of A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, N). On entry, the matrix B in the pair (A, B). On exit, B has been overwritten.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B. LDB >= max(1, N).

ALPHA Output parameter.

ALPHA is COMPLEX

ALPHA is an array, dimension (N) .

BETA Output parameter.

BETA is COMPLEX

BETA is an array, dimension (N). On exit, ALPHA(j)/BETA(j), j=1,..., N, will be the generalized eigenvalues.

Note: the quotients ALPHA(j)/BETA(j) may easily over- or underflow, and BETA(j) may even be zero. Thus, the user should avoid naively computing the ratio alpha/beta. However, ALPHA will be always less than and usually comparable with norm(A) in magnitude, and BETA always less than and usually comparable with norm(B).

VL Output parameter.

VL is COMPLEX

VL is an array, dimension (LDVL, N). If JOBVL = 'V', the left generalized eigenvectors u(j) are stored one after another in the columns of VL, in the same order as their eigenvalues. Each eigenvector is scaled so the largest component has abs(real part) + abs(imag. part) = 1. Not referenced if JOBVL = 'N'.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the matrix VL. LDVL >= 1, and if JOBVL = 'V', LDVL >= N.

VR Output parameter.

VR is COMPLEX

VR is an array, dimension (LDVR, N). If JOBVR = 'V', the right generalized eigenvectors v(j) are stored one after another in the columns of VR, in the same order as their eigenvalues. Each eigenvector is scaled so the largest component has abs(real part) + abs(imag. part) = 1. Not referenced if JOBVR = 'N'.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the matrix VR. LDVR >= 1, and if JOBVR = 'V', LDVR >= N.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (8*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. =1,. . . ,N: The QZ iteration failed. No eigenvectors have been calculated, but ALPHA(j) and BETA(j) should be correct for j=INFO+1,. . . ,N. > N: =N+1: other then QZ iteration failed in SHGEQZ, =N+2: error return from STGEVC.

Related Information

For this routine in other precisions, please see *dggev3*, *sggev3* and *zggev3*. It also exists with a native C interface as *LAPACKE_cggev3*.

4.14.8 cggevx

cggevx computes for a pair of N-by-N complex nonsymmetric matrices (A,B) the generalized eigenvalues, and optionally, the left and/or right generalized eigenvectors.

Optionally, it also computes a balancing transformation to improve the conditioning of the eigenvalues and eigenvectors (ILO, IHI, LSCALE, RSCALE, ABNRM, and BBNRM), reciprocal condition numbers for the eigenvalues (RCONDE), and reciprocal condition numbers for the right eigenvectors (RCONDV).

A generalized eigenvalue for a pair of matrices (A,B) is a scalar lambda or a ratio alpha/beta = lambda, such that A - lambda*B is singular. It is usually represented as the pair (alpha,beta), as there is a reasonable interpretation for beta=0, and even for both being zero.

The right eigenvector v(j) corresponding to the eigenvalue lambda(j) of (A,B) satisfies

```
A * v(j) = lambda(j) * B * v(j) .
```

The left eigenvector u(j) corresponding to the eigenvalue lambda(j) of (A,B) satisfies

```
u(j)**H * A  = lambda(j) * u(j)**H * B.
```

where $u(j)^H$ is the conjugate-transpose of u(j).

Syntax

Fortran specification:

```
use armpl_library

subroutine cggevx(BALANC, JOBVL, JOBVR, SENSE, N, A, LDA, B, LDB, ALPHA, BETA,
                  VL, LDVL, VR, LDVR, ILO, IHI, LSCALE, RSCALE, ABNRM, BBNRM,
                  RCONDE, RCONDV, WORK, LWORK, RWORK, IWORK, BWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cggevx_(const char *balanc, const char *jobvl, const char *jobvr,
             const char *sense, const armpl_int_t *n,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             armpl_singlecomplex_t *b, const armpl_int_t *ldb,
             armpl_singlecomplex_t *alpha, armpl_singlecomplex_t *beta,
             armpl_singlecomplex_t *vl, const armpl_int_t *ldvl,
             armpl_singlecomplex_t *vr, const armpl_int_t *ldvr,
             armpl_int_t *ilo, armpl_int_t *ihi, float *lscale, float *rscale,
             float *abnrm, float *bbnrm, float *rconde, float *rcondv,
             armpl_singlecomplex_t *work, const armpl_int_t *lwork,
             float *rwork, armpl_int_t *iwork, armpl_int_t *bwork,
             armpl_int_t *info, ... );
```

Parameters

BALANC Input parameter.

BALANC is CHARACTER*1

Specifies the balance option to be performed: = 'N': do not diagonally scale or permute; = 'P': permute only; = 'S': scale only; = 'B': both permute and scale. Computed reciprocal condition numbers will be for the matrices after permuting and/or balancing. Permuting does not change condition numbers (in exact arithmetic), but balancing does.

JOBVL Input parameter.

JOBVL is CHARACTER*1

= 'N': do not compute the left generalized eigenvectors; = 'V': compute the left generalized eigenvectors.

JOBVR Input parameter.

JOBVR is CHARACTER*1

= 'N': do not compute the right generalized eigenvectors; = 'V': compute the right generalized eigenvectors.

SENSE Input parameter.

SENSE is CHARACTER*1

Determines which reciprocal condition numbers are computed. = 'N': none are computed; = 'E': computed for eigenvalues only; = 'V': computed for eigenvectors only; = 'B': computed for eigenvalues and eigenvectors.

N Input parameter.

N is INTEGER

The order of the matrices A, B, VL, and VR. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the matrix A in the pair (A, B). On exit, A has been overwritten. If JOBVL='V' or JOBVR='V' or both, then A contains the first part of the complex Schur form of the "balanced" versions of the input A and B.

LDA Input parameter.

LDA is INTEGER

The leading dimension of A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, N). On entry, the matrix B in the pair (A, B). On exit, B has been overwritten. If JOBVL='V' or JOBVR='V' or both, then B contains the second part of the complex Schur form of the "balanced" versions of the input A and B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B. LDB >= max(1, N).

ALPHA Output parameter.

ALPHA is COMPLEX

ALPHA is an array, dimension (N) .

BETA Output parameter.

BETA is COMPLEX

BETA is an array, dimension (N). On exit, ALPHA(j)/BETA(j), j=1,..., N, will be the generalized eigenvalues.

Note: the quotient ALPHA(j)/BETA(j)) may easily over- or underflow, and BETA(j) may even be zero. Thus, the user should avoid naively computing the ratio ALPHA/BETA. However, ALPHA will be always less than and usually comparable with norm(A) in magnitude, and BETA always less than and usually comparable with norm(B).

VL Output parameter.

VL is COMPLEX

VL is an array, dimension (LDVL, N). If JOBVL = 'V', the left generalized eigenvectors u(j) are stored one after another in the columns of VL, in the same order as their eigenvalues. Each eigenvector will be scaled so the largest component will have abs(real part) + abs(imag. part) = 1. Not referenced if JOBVL = 'N'.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the matrix VL. LDVL >= 1, and if JOBVL = 'V', LDVL >= N.

VR Output parameter.

VR is COMPLEX

VR is an array, dimension (LDVR, N). If JOBVR = 'V', the right generalized eigenvectors v(j) are stored one after another in the columns of VR, in the same order as their eigenvalues. Each eigenvector will be scaled so the largest component will have abs(real part) + abs(imag. part) = 1. Not referenced if JOBVR = 'N'.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the matrix VR. LDVR >= 1, and if JOBVR = 'V', LDVR >= N.

ILO Output parameter.

ILO is INTEGER

IHI Output parameter.

IHI is INTEGER

ILO and IHI are integer values such that on exit A(i,j) = 0 and B(i,j) = 0 if i > j and j = 1,...,ILO-1 or i = IHI+1,...,N. If BALANC = 'N' or 'S', ILO = 1 and IHI = N.

LSCALE Output parameter.

LSCALE is REAL

LSCALE is an array, dimension (N). Details of the permutations and scaling factors applied to the left side of A and B. If PL(j) is the index of the row interchanged with row j, and DL(j) is the scaling factor applied to row j, then LSCALE(j) = PL(j) for j = 1,...,ILO-1 = DL(j) for j = ILO,...,IHI = PL(j) for j = IHI+1,...,N. The order in which the interchanges are made is N to IHI+1, then 1 to ILO-1.

RSCALE Output parameter.

RSCALE is REAL

RSCALE is an array, dimension (N). Details of the permutations and scaling factors applied to the right side of A and B. If PR(j) is the index of the column interchanged with column j, and DR(j) is the scaling factor applied to column j, then RSCALE(j) = PR(j) for j = 1,...,ILO-1 = DR(j) for j = ILO,...,IHI = PR(j) for j = IHI+1,...,N The order in which the interchanges are made is N to IHI+1, then 1 to ILO-1.

ABNRM Output parameter.

ABNRM is REAL

The one-norm of the balanced matrix A.

BBNRM Output parameter.

BBNRM is REAL

The one-norm of the balanced matrix B.

RCONDE Output parameter.

RCONDE is REAL

RCONDE is an array, dimension (N). If SENSE = 'E' or 'B', the reciprocal condition numbers of the eigenvalues, stored in consecutive elements of the array. If SENSE = 'N' or 'V', RCONDE is not referenced.

RCONDV Output parameter.

RCONDV is REAL

RCONDV is an array, dimension (N). If SENSE = 'V' or 'B', the estimated reciprocal condition numbers of the eigenvectors, stored in consecutive elements of the array. If the eigenvalues cannot be reordered to compute RCONDV(j), RCONDV(j) is set to 0; this can only occur when the true value would be very small anyway. If SENSE = 'N' or 'E', RCONDV is not referenced.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,2*N). If SENSE = 'E', LWORK >= max(1,4*N). If SENSE = 'V' or 'B', LWORK >= max(1,2*N*N+2*N).

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (lrwork). lrwork must be at least max(1,6*N) if BALANC = 'S' or 'B', and at least max(1,2*N) otherwise. Real workspace.

IWORK Output parameter.

IWORK is INTEGER array, dimension (N+2)

If SENSE = 'E', IWORK is not referenced.

BWORK Output parameter.

BWORK is LOGICAL

BWORK is an array, dimension (N). If SENSE = 'N', BWORK is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. = 1,...,N: The QZ iteration failed. No eigenvectors have been calculated, but ALPHA(j) and BETA(j) should be correct for j=INFO+1,...,N. > N: =N+1: other than QZ iteration failed in CHGEQZ. =N+2: error return from CTGEVC.

Related Information

For this routine in other precisions, please see *dggevx*, *sggevx* and *zggevx*. It also exists with a native C interface as *LAPACKE_cggevx*.

4.14.9 cgghd3

cgghd3 reduces a pair of complex matrices (A,B) to generalized upper Hessenberg form using unitary transformations, where A is a general matrix and B is upper triangular. The form of the generalized eigenvalue problem is

```
A*x = lambda*B*x,
```

and B is typically made upper triangular by computing its QR factorization and moving the unitary matrix Q to the left side of the equation.

This subroutine simultaneously reduces A to a Hessenberg matrix H:

```
Q**H*A*Z = H
```

and transforms B to another upper triangular matrix T:

```
Q**H*B*Z = T
```

in order to reduce the problem to its standard form

```
H*y = lambda*T*y
```

where y = Z^H *x.

The unitary matrices Q and Z are determined as products of Givens rotations. They may either be formed explicitly, or they may be postmultiplied into input matrices Q1 and Z1, so that

```
 Q1 * A * Z1**H = (Q1*Q) * H * (Z1*Z)**H

Q1 * B * Z1**H = (Q1*Q) * T * (Z1*Z)**H
```

If Q1 is the unitary matrix from the QR factorization of B in the original equation A*x = lambda*B*x, then `cgghd3` reduces the original problem to generalized Hessenberg form.

This is a blocked variant of CGGHRD, using matrix-matrix multiplications for parts of the computation to enhance performance.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgghd3(COMPQ, COMPZ, N, ILO, IHI, A, LDA, B, LDB, Q, LDQ, Z, LDZ,
                  WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgghd3_(const char *compq, const char *compz, const armpl_int_t *n,
             const armpl_int_t *ilo, const armpl_int_t *ihi,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             armpl_singlecomplex_t *b, const armpl_int_t *ldb,
             armpl_singlecomplex_t *q, const armpl_int_t *ldq,
             armpl_singlecomplex_t *z, const armpl_int_t *ldz,
             armpl_singlecomplex_t *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

COMPQ Input parameter.

`COMPQ` is `CHARACTER*1`

= 'N': do not compute Q; = 'I': `Q` is initialized to the unit matrix, and the unitary matrix `Q` is returned; = 'V': `Q` must contain a unitary matrix Q1 on entry, and the product Q1* `Q` is returned.

COMPZ Input parameter.

COMPZ is CHARACTER*1

= 'N': do not compute Z; = 'I': Z is initialized to the unit matrix, and the unitary matrix Z is returned; = 'V': Z must contain a unitary matrix Z1 on entry, and the product Z1* Z is returned.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

ILO and IHI mark the rows and columns of A which are to be reduced. It is assumed that A is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N. ILO and IHI are normally set by a previous call to CGGBAL; otherwise they should be set to 1 and N respectively. 1 <= ILO <= IHI <= N, if N > 0; ILO=1 and IHI=0, if N=0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the N-by-N general matrix to be reduced. On exit, the upper triangle and the first subdiagonal of A are overwritten with the upper Hessenberg matrix H, and the rest is set to zero.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, N). On entry, the N-by-N upper triangular matrix B. On exit, the upper triangular matrix T = Q^H B Z. The elements below the diagonal are set to zero.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Q Input and output parameter.

Q is COMPLEX

Q is an array, dimension (LDQ, N). On entry, if COMPQ = 'V', the unitary matrix Q1, typically from the QR factorization of B. On exit, if COMPQ='I', the unitary matrix Q, and if COMPQ = 'V', the product Q1*Q. Not referenced if COMPQ='N'.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= N if COMPQ='V' or 'I'; LDQ >= 1 otherwise.

Z Input and output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, N). On entry, if COMPZ = 'V', the unitary matrix Z1. On exit, if COMPZ='I', the unitary matrix Z, and if COMPZ = 'V', the product Z1*Z. Not referenced if COMPZ='N'.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= N if COMPZ='V' or 'I'; LDZ >= 1 otherwise.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (LWORK). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= 1. For optimum performance LWORK >= 6*N*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *dgghd3*, *sgghd3* and *zgghd3*. It also exists with a native C interface as *LAPACKE_cgghd3*.

4.14.10 cgghrd

cgghrd reduces a pair of complex matrices (A,B) to generalized upper Hessenberg form using unitary transformations, where A is a general matrix and B is upper triangular. The form of the generalized eigenvalue problem is

```
A*x = lambda*B*x,
```

and B is typically made upper triangular by computing its QR factorization and moving the unitary matrix Q to the left side of the equation.

This subroutine simultaneously reduces A to a Hessenberg matrix H:

```
Q**H*A*Z = H
```

and transforms B to another upper triangular matrix T:

```
Q**H*B*Z = T
```

in order to reduce the problem to its standard form

```
H*y = lambda*T*y
```

where y = Z^H *x.

The unitary matrices Q and Z are determined as products of Givens rotations. They may either be formed explicitly, or they may be postmultiplied into input matrices Q1 and Z1, so that

```
 Q1 * A * Z1**H = (Q1*Q) * H * (Z1*Z)**H
Q1 * B * Z1**H = (Q1*Q) * T * (Z1*Z)**H
```

If Q1 is the unitary matrix from the QR factorization of B in the original equation A*x = lambda*B*x, then `cgghrd` reduces the original problem to generalized Hessenberg form.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgghrd(COMPQ, COMPZ, N, ILO, IHI, A, LDA, B, LDB, Q, LDQ, Z, LDZ,
                  INFO)
```

C specification:

```
#include "armpl.h"

void cgghrd_(const char *compq, const char *compz, const armpl_int_t *n,
             const armpl_int_t *ilo, const armpl_int_t *ihi,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             armpl_singlecomplex_t *b, const armpl_int_t *ldb,
             armpl_singlecomplex_t *q, const armpl_int_t *ldq,
             armpl_singlecomplex_t *z, const armpl_int_t *ldz,
             armpl_int_t *info, ... );
```

Parameters

COMPQ Input parameter.

COMPQ is CHARACTER*1

= 'N': do not compute Q; = 'I': Q is initialized to the unit matrix, and the unitary matrix Q is returned; = 'V': Q must contain a unitary matrix Q1 on entry, and the product Q1* Q is returned.

COMPZ Input parameter.

COMPZ is CHARACTER*1

= 'N': do not compute Z; = 'I': Z is initialized to the unit matrix, and the unitary matrix Z is returned; = 'V': Z must contain a unitary matrix Z1 on entry, and the product Z1* Z is returned.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

ILO and IHI mark the rows and columns of A which are to be reduced. It is assumed that A is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N. ILO and IHI are normally set by a previous call to CGGBAL; otherwise they should be set to 1 and N respectively. 1 <= ILO <= IHI <= N, if N > 0; ILO=1 and IHI=0, if N=0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the N-by-N general matrix to be reduced. On exit, the upper triangle and the first subdiagonal of A are overwritten with the upper Hessenberg matrix H, and the rest is set to zero.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, N). On entry, the N-by-N upper triangular matrix B. On exit, the upper triangular matrix T = Q^H B Z. The elements below the diagonal are set to zero.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Q Input and output parameter.

Q is COMPLEX

Q is an array, dimension (LDQ, N). On entry, if COMPQ = 'V', the unitary matrix Q1, typically from the QR factorization of B. On exit, if COMPQ='I', the unitary matrix Q, and if COMPQ = 'V', the product Q1*Q. Not referenced if COMPQ='N'.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= N if COMPQ='V' or 'I'; LDQ >= 1 otherwise.

Z Input and output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, N). On entry, if COMPZ = 'V', the unitary matrix Z1. On exit, if COMPZ='I', the unitary matrix Z, and if COMPZ = 'V', the product Z1*Z. Not referenced if COMPZ='N'.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= N if COMPZ='V' or 'I'; LDZ >= 1 otherwise.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *dgghrd*, *sgghrd* and *zgghrd*. It also exists with a native C interface as *LAPACKE_cgghrd*.

4.14.11 chgeqz

chgeqz computes the eigenvalues of a complex matrix pair (H,T), where H is an upper Hessenberg matrix and T is upper triangular, using the single-shift QZ method. Matrix pairs of this type are produced by the reduction to generalized upper Hessenberg form of a complex matrix pair (A,B):

```
A = Q1*H*Z1**H,  B = Q1*T*Z1**H,
```

as computed by CGGHRD.

If JOB='S', then the Hessenberg-triangular pair (H,T) is also reduced to generalized Schur form,

```
H = Q*S*Z**H,  T = Q*P*Z**H,
```

where Q and Z are unitary matrices and S and P are upper triangular.

Optionally, the unitary matrix Q from the generalized Schur factorization may be postmultiplied into an input matrix Q1, and the unitary matrix Z may be postmultiplied into an input matrix Z1. If Q1 and Z1 are the unitary matrices from CGGHRD that reduced the matrix pair (A,B) to generalized Hessenberg form, then the output matrices Q1*Q and Z1*Z are the unitary factors from the generalized Schur factorization of (A,B):

```
A = (Q1*Q)*S*(Z1*Z)**H,  B = (Q1*Q)*P*(Z1*Z)**H.
```

To avoid overflow, eigenvalues of the matrix pair (H,T) (equivalently, of (A,B)) are computed as a pair of complex values (alpha,beta). If beta is nonzero, lambda = alpha / beta is an eigenvalue of the generalized nonsymmetric eigenvalue problem (GNEP)

```
A*x = lambda*B*x
```

and if alpha is nonzero, mu = beta / alpha is an eigenvalue of the alternate form of the GNEP

```
mu*A*y = B*y.
```

The values of alpha and beta for the i-th eigenvalue can be read directly from the generalized Schur form: alpha = S(i,i), beta = P(i,i).

Ref: C.B. Moler & G.W. Stewart, “An Algorithm for Generalized Matrix Eigenvalue Problems”, SIAM J. Numer. Anal., 10(1973), pp. 241–256.

Syntax

Fortran specification:

```
use armpl_library

subroutine chgeqz(JOB, COMPQ, COMPZ, N, ILO, IHI, H, LDH, T, LDT, ALPHA, BETA,
                  Q, LDQ, Z, LDZ, WORK, LWORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void chgeqz_(const char *job, const char *compq, const char *compz,
             const armpl_int_t *n, const armpl_int_t *ilo,
             const armpl_int_t *ihi, armpl_singlecomplex_t *h,
             const armpl_int_t *ldh, armpl_singlecomplex_t *t,
             const armpl_int_t *ldt, armpl_singlecomplex_t *alpha,
             armpl_singlecomplex_t *beta, armpl_singlecomplex_t *q,
             const armpl_int_t *ldq, armpl_singlecomplex_t *z,
             const armpl_int_t *ldz, armpl_singlecomplex_t *work,
             const armpl_int_t *lwork, float *rwork, armpl_int_t *info,
             ... );
```

Parameters

JOB Input parameter.

JOB is CHARACTER*1

= ‘E’: Compute eigenvalues only; = ‘S’: Computer eigenvalues and the Schur form.

COMPQ Input parameter.

COMPQ is CHARACTER*1

= 'N': Left Schur vectors (Q) are not computed; = 'I': Q is initialized to the unit matrix and the matrix Q of left Schur vectors of (H, T) is returned; = 'V': Q must contain a unitary matrix Q1 on entry and the product Q1* Q is returned.

COMPZ Input parameter.

COMPZ is CHARACTER*1

= 'N': Right Schur vectors (Z) are not computed; = 'I': Q is initialized to the unit matrix and the matrix Z of right Schur vectors of (H, T) is returned; = 'V': Z must contain a unitary matrix Z1 on entry and the product Z1* Z is returned.

N Input parameter.

N is INTEGER

The order of the matrices H, T, Q, and Z. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

ILO and IHI mark the rows and columns of H which are in Hessenberg form. It is assumed that A is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N. If N > 0, 1 <= ILO <= IHI <= N; if N = 0, ILO=1 and IHI=0.

H Input and output parameter.

H is COMPLEX

H is an array, dimension (LDH, N). On entry, the N-by-N upper Hessenberg matrix H. On exit, if JOB = 'S', H contains the upper triangular matrix S from the generalized Schur factorization. If JOB = 'E', the diagonal of H matches that of S, but the rest of H is unspecified.

LDH Input parameter.

LDH is INTEGER

The leading dimension of the array H. LDH >= max(1, N).

T Input and output parameter.

T is COMPLEX

T is an array, dimension (LDT, N). On entry, the N-by-N upper triangular matrix T. On exit, if JOB = 'S', T contains the upper triangular matrix P from the generalized Schur factorization. If JOB = 'E', the diagonal of T matches that of P, but the rest of T is unspecified.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

ALPHA Output parameter.

ALPHA is COMPLEX

ALPHA is an array, dimension (N). The complex scalars alpha that define the eigenvalues of GNEP. ALPHA(i) = S(i,i) in the generalized Schur factorization.

BETA Output parameter.

BETA is COMPLEX

BETA is an array, dimension (N). The real non-negative scalars beta that define the eigenvalues of GNEP. BETA(i) = P(i,i) in the generalized Schur factorization.

Together, the quantities alpha = ALPHA(j) and beta = BETA(j) represent the j-th eigenvalue of the matrix pair (A,B), in one of the forms lambda = alpha/beta or mu = beta/alpha. Since either lambda or mu may overflow, they should not, in general, be computed.

Q Input and output parameter.

Q is COMPLEX

Q is an array, dimension (LDQ, N). On entry, if COMPQ = 'V', the unitary matrix Q1 used in the reduction of (A,B) to generalized Hessenberg form. On exit, if COMPQ = 'I', the unitary matrix of left Schur vectors of (H, T), and if COMPQ = 'V', the unitary matrix of left Schur vectors of (A,B). Not referenced if COMPQ = 'N'.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= 1. If COMPQ='V' or 'I', then LDQ >= N.

Z Input and output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, N). On entry, if COMPZ = 'V', the unitary matrix Z1 used in the reduction of (A,B) to generalized Hessenberg form. On exit, if COMPZ = 'I', the unitary matrix of right Schur vectors of (H, T), and if COMPZ = 'V', the unitary matrix of right Schur vectors of (A,B). Not referenced if COMPZ = 'N'.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1. If COMPZ='V' or 'I', then LDZ >= N.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO >= 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N).

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value = 1,...,N: the QZ iteration did not converge. (H, T) is not in Schur form, but ALPHA(i) and BETA(i), i=INFO+1,...,N should be correct. = N+1,...,2*N: the shift calculation failed. (H, T) is not in Schur form, but ALPHA(i) and BETA(i), i=INFO-N+1,...,N should be correct.

Related Information

For this routine in other precisions, please see *dhgeqz*, *shgeqz* and *zhgeqz*. It also exists with a native C interface as *LAPACKE_chgeqz*.

4.14.12 ctgevc

`ctgevc` computes some or all of the right and/or left eigenvectors of a pair of complex matrices (S,P), where S and P are upper triangular. Matrix pairs of this type are produced by the generalized Schur factorization of a complex matrix pair (A,B):

```
A = Q*S*Z**H,  B = Q*P*Z**H
```

as computed by CGGHRD + CHGEQZ.

The right eigenvector x and the left eigenvector y of (S,P) corresponding to an eigenvalue w are defined by:

```
S*x = w*P*x,  (y**H)*S = w*(y**H)*P,
```

where y^H denotes the conjugate tranpose of y. The eigenvalues are not input to this routine, but are computed directly from the diagonal elements of S and P.

This routine returns the matrices X and/or Y of right and left eigenvectors of (S,P), or the products Z*X and/or Q*Y, where Z and Q are input matrices. If Q and Z are the unitary factors from the generalized Schur factorization of a matrix pair (A,B), then Z*X and Q*Y are the matrices of right and left eigenvectors of (A,B).

Syntax

Fortran specification:

```
use armpl_library

subroutine ctgevc(SIDE, HOWMNY, SELECT, N, S, LDS, P, LDP, VL, LDVL, VR, LDVR,
                  MM, M, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ctgevc_(const char *side, const char *howmny, const armpl_int_t *select,
             const armpl_int_t *n, const armpl_singlecomplex_t *s,
             const armpl_int_t *lds, const armpl_singlecomplex_t *p,
             const armpl_int_t *ldp, armpl_singlecomplex_t *vl,
             const armpl_int_t *ldvl, armpl_singlecomplex_t *vr,
             const armpl_int_t *ldvr, const armpl_int_t *mm, armpl_int_t *m,
             armpl_singlecomplex_t *work, float *rwork, armpl_int_t *info,
             ... );
```

Parameters

SIDE Input parameter.

`SIDE` is `CHARACTER*1`

= 'R': compute right eigenvectors only; = 'L': compute left eigenvectors only; = 'B': compute both right and left eigenvectors.

HOWMNY Input parameter.

`HOWMNY` is `CHARACTER*1`

= ‘A’: compute all right and/or left eigenvectors; = ‘B’: compute all right and/or left eigenvectors, backtransformed by the matrices in VR and/or VL; = ‘S’: compute selected right and/or left eigenvectors, specified by the logical array SELECT.

SELECT Input parameter.

SELECT is LOGICAL

SELECT is an array, dimension (N). If HOWMNY=’S’, SELECT specifies the eigenvectors to be computed. The eigenvector corresponding to the j-th eigenvalue is computed if SELECT(j) = .TRUE.. Not referenced if HOWMNY = ‘A’ or ‘B’.

N Input parameter.

N is INTEGER

The order of the matrices S and P. N >= 0.

S Input parameter.

S is COMPLEX

S is an array, dimension (LDS, N). The upper triangular matrix S from a generalized Schur factorization, as computed by CHGEQZ.

LDS Input parameter.

LDS is INTEGER

The leading dimension of array S. LDS >= max(1, N).

P Input parameter.

P is COMPLEX

P is an array, dimension (LDP, N). The upper triangular matrix P from a generalized Schur factorization, as computed by CHGEQZ. P must have real diagonal elements.

LDP Input parameter.

LDP is INTEGER

The leading dimension of array P. LDP >= max(1, N).

VL Input and output parameter.

VL is COMPLEX

VL is an array, dimension (LDVL, MM). On entry, if SIDE = ‘L’ or ‘B’ and HOWMNY = ‘B’, VL must contain an N-by-N matrix Q (usually the unitary matrix Q of left Schur vectors returned by CHGEQZ). On exit, if SIDE = ‘L’ or ‘B’, VL contains: if HOWMNY = ‘A’, the matrix Y of left eigenvectors of (S, P); if HOWMNY = ‘B’, the matrix Q*Y; if HOWMNY = ‘S’, the left eigenvectors of (S, P) specified by SELECT, stored consecutively in the columns of VL, in the same order as their eigenvalues. Not referenced if SIDE = ‘R’.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of array VL. LDVL >= 1, and if SIDE = ‘L’ or ‘l’ or ‘B’ or ‘b’, LDVL >= N.

VR Input and output parameter.

VR is COMPLEX

VR is an array, dimension (LDVR, MM). On entry, if SIDE = ‘R’ or ‘B’ and HOWMNY = ‘B’, VR must contain an N-by-N matrix Q (usually the unitary matrix Z of right Schur vectors returned by CHGEQZ). On exit, if SIDE = ‘R’ or ‘B’, VR contains: if HOWMNY = ‘A’, the matrix X of right eigenvectors of (S, P); if HOWMNY = ‘B’, the matrix Z*X; if HOWMNY = ‘S’, the right eigenvectors of (S, P) specified by SELECT, stored consecutively in the columns of VR, in the same order as their eigenvalues. Not referenced if SIDE = ‘L’.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the array VR. LDVR >= 1, and if SIDE = 'R' or 'B', LDVR >= N.

MM Input parameter.

MM is INTEGER

The number of columns in the arrays VL and/or VR. MM >= M.

M Output parameter.

M is INTEGER

The number of columns in the arrays VL and/or VR actually used to store the eigenvectors. If HOWMNY = 'A' or 'B', M is set to N. Each selected eigenvector occupies one column.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *dtgevc*, *stgevc* and *ztgevc*. It also exists with a native C interface as *LAPACKE_ctgevc*.

4.14.13 ctgexc

ctgexc reorders the generalized Schur decomposition of a complex matrix pair (A,B), using an unitary equivalence transformation (A, B) := Q * (A, B) * Z^H , so that the diagonal block of (A, B) with row index IFST is moved to row ILST.

(A, B) must be in generalized Schur canonical form, that is, A and B are both upper triangular.

Optionally, the matrices Q and Z of generalized Schur vectors are updated.

```
Q(in) * A(in) * Z(in)**H = Q(out) * A(out) * Z(out)**H
Q(in) * B(in) * Z(in)**H = Q(out) * B(out) * Z(out)**H
```

Syntax

Fortran specification:

```
use armpl_library

subroutine ctgexc(WANTQ, WANTZ, N, A, LDA, B, LDB, Q, LDQ, Z, LDZ, IFST, ILST,
                  INFO)
```

C specification:

```
#include "armpl.h"

void ctgexc_(const armpl_int_t *wantq, const armpl_int_t *wantz,
             const armpl_int_t *n, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, armpl_singlecomplex_t *q,
             const armpl_int_t *ldq, armpl_singlecomplex_t *z,
             const armpl_int_t *ldz, const armpl_int_t *ifst,
             armpl_int_t *ilst, armpl_int_t *info);
```

Parameters

WANTQ Input parameter.

WANTQ is LOGICAL

.TRUE. : update the left transformation matrix Q; .FALSE.: do not update Q.

WANTZ Input parameter.

WANTZ is LOGICAL

.TRUE. : update the right transformation matrix Z; .FALSE.: do not update Z.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the upper triangular matrix A in the pair (A, B). On exit, the updated matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, N). On entry, the upper triangular matrix B in the pair (A, B). On exit, the updated matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Q Input and output parameter.

Q is COMPLEX

Q is an array, dimension (LDQ, N). On entry, if WANTQ = .TRUE., the unitary matrix Q. On exit, the updated matrix Q. If WANTQ = .FALSE., Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= 1; If WANTQ = .TRUE., LDQ >= N.

Z Input and output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, N). On entry, if WANTZ = .TRUE., the unitary matrix Z. On exit, the updated matrix Z. If WANTZ = .FALSE., Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1; If WANTZ = .TRUE., LDZ >= N.

IFST Input parameter.

IFST is INTEGER

ILST Input and output parameter.

ILST is INTEGER

Specify the reordering of the diagonal blocks of (A, B). The block with row index IFST is moved to row ILST, by a sequence of swapping between adjacent blocks.

INFO Output parameter.

INFO is INTEGER

=0: Successful exit. <0: if INFO = -i, the i-th argument had an illegal value. =1: The transformed matrix pair (A, B) would be too far from generalized Schur form; the problem is ill- conditioned. (A, B) may have been partially reordered, and ILST points to the first row of the current position of the block being moved.

Related Information

For this routine in other precisions, please see *dtgexc*, *stgexc* and *ztgexc*. It also exists with a native C interface as *LAPACKE_ctgexc*.

4.14.14 ctgsen

ctgsen reorders the generalized Schur decomposition of a complex matrix pair (A, B) (in terms of an unitary equivalence trans- formation Q^H * (A, B) * Z), so that a selected cluster of eigenvalues appears in the leading diagonal blocks of the pair (A,B). The leading columns of Q and Z form unitary bases of the corresponding left and right eigenspaces (deflating subspaces). (A, B) must be in generalized Schur canonical form, that is, A and B are both upper triangular.

ctgsen also computes the generalized eigenvalues

```
w(j)= ALPHA(j) / BETA(j)
```

of the reordered matrix pair (A, B).

Optionally, the routine computes estimates of reciprocal condition numbers for eigenvalues and eigenspaces. These are Difu[(A11,B11), (A22,B22)] and Difl[(A11,B11), (A22,B22)], i.e. the separation(s) between the matrix pairs (A11, B11) and (A22,B22) that correspond to the selected cluster and the eigenvalues outside the cluster, resp., and norms of "projections" onto left and right eigenspaces w.r.t. the selected cluster in the (1,1)-block.

Syntax

Fortran specification:

```
use armpl_library

subroutine ctgsen(IJOB, WANTQ, WANTZ, SELECT, N, A, LDA, B, LDB, ALPHA, BETA,
                  Q, LDQ, Z, LDZ, M, PL, PR, DIF, WORK, LWORK, IWORK, LIWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void ctgsen_(const armpl_int_t *ijob, const armpl_int_t *wantq,
             const armpl_int_t *wantz, const armpl_int_t *select,
             const armpl_int_t *n, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, armpl_singlecomplex_t *alpha,
             armpl_singlecomplex_t *beta, armpl_singlecomplex_t *q,
             const armpl_int_t *ldq, armpl_singlecomplex_t *z,
             const armpl_int_t *ldz, armpl_int_t *m, float *pl, float *pr,
             float *dif, armpl_singlecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info);
```

Parameters

IJOB Input parameter.

IJOB is INTEGER

Specifies whether condition numbers are required for the cluster of eigenvalues (PL and PR) or the deflating subspaces (Difu and Difl): =0: Only reorder w.r.t. SELECT. No extras. =1: Reciprocal of norms of "projections" onto left and right eigenspaces w.r.t. the selected cluster (PL and PR). =2: Upper bounds on Difu and Difl. F-norm-based estimate (DIF(1:2)). =3: Estimate of Difu and Difl. 1-norm-based estimate (DIF(1:2)). About 5 times as expensive as IJOB = 2. =4: Compute PL, PR and DIF (i.e. 0, 1 and 2 above): Economic version to get it all. =5: Compute PL, PR and DIF (i.e. 0, 1 and 3 above)

WANTQ Input parameter.

WANTQ is LOGICAL

.TRUE. : update the left transformation matrix Q; .FALSE.: do not update Q.

WANTZ Input parameter.

WANTZ is LOGICAL

.TRUE. : update the right transformation matrix Z; .FALSE.: do not update Z.

SELECT Input parameter.

SELECT is LOGICAL

SELECT is an array, dimension (N). SELECT specifies the eigenvalues in the selected cluster. To select an eigenvalue w(j), SELECT(j) must be set to .TRUE..

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension(LDA, N). On entry, the upper triangular matrix A, in generalized Schur canonical form. On exit, A is overwritten by the reordered matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX

B is an array, dimension(LDB, N). On entry, the upper triangular matrix B, in generalized Schur canonical form. On exit, B is overwritten by the reordered matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

ALPHA Output parameter.

ALPHA is COMPLEX

ALPHA is an array, dimension (N) .

BETA Output parameter.

BETA is COMPLEX

BETA is an array, dimension (N). The diagonal elements of A and B, respectively, when the pair (A, B) has been reduced to generalized Schur form. ALPHA(i)/BETA(i) i=1,. . . ,N are the generalized eigenvalues.

Q Input and output parameter.

Q is COMPLEX

Q is an array, dimension (LDQ, N). On entry, if WANTQ = .TRUE., Q is an N-by-N matrix. On exit, Q has been postmultiplied by the left unitary transformation matrix which reorder (A, B); The leading M columns of Q form orthonormal bases for the specified pair of left eigenspaces (deflating subspaces). If WANTQ = .FALSE., Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= 1. If WANTQ = .TRUE., LDQ >= N.

Z Input and output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, N). On entry, if WANTZ = .TRUE., Z is an N-by-N matrix. On exit, Z has been postmultiplied by the left unitary transformation matrix which reorder (A, B); The leading M columns of Z form orthonormal bases for the specified pair of left eigenspaces (deflating subspaces). If WANTZ = .FALSE., Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1. If WANTZ = .TRUE., LDZ >= N.

M Output parameter.

M is INTEGER

The dimension of the specified pair of left and right eigenspaces, (deflating subspaces) 0 <= M <= N.

PL Output parameter.

PL is REAL

PR Output parameter.

PR is REAL

If IJOB = 1, 4 or 5, PL, PR are lower bounds on the reciprocal of the norm of "projections" onto left and right eigenspace with respect to the selected cluster. 0 < PL, PR <= 1. If M = 0 or M = N, PL = PR = 1. If IJOB = 0, 2 or 3 PL, PR are not referenced.

DIF Output parameter.

DIF is REAL

DIF is an array, dimension (2).. If IJOB >= 2, DIF(1:2) store the estimates of Difu and Difl. If IJOB = 2 or 4, DIF(1:2) are F-norm-based upper bounds on Difu and Difl. If IJOB = 3 or 5, DIF(1:2) are 1-norm-based estimates of Difu and Difl, computed using reversed communication with CLACN2. If M = 0 or N, DIF(1:2) = F-norm([A, B]). If IJOB = 0 or 1, DIF is not referenced.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= 1 If IJOB = 1, 2 or 4, LWORK >= 2*M*(N-M) If IJOB = 3 or 5, LWORK >= 4*M*(N-M)

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. LIWORK >= 1. If IJOB = 1, 2 or 4, LIWORK >= N+2; If IJOB = 3 or 5, LIWORK >= MAX(N+2, 2*M*(N-M));

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the IWORK array, returns this value as the first entry of the IWORK array, and no error message related to LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

=0: Successful exit. <0: If INFO = -i, the i-th argument had an illegal value. =1: Reordering of (A, B) failed because the transformed matrix pair (A, B) would be too far from generalized Schur form; the problem is very ill-conditioned. (A, B) may have been partially reordered. If requested, 0 is returned in DIF(*), PL and PR.

Related Information

For this routine in other precisions, please see *dtgsen*, *stgsen* and *ztgsen*. It also exists with a native C interface as *LAPACKE_ctgsen*.

4.14.15 ctgsna

ctgsna estimates reciprocal condition numbers for specified eigenvalues and/or eigenvectors of a matrix pair (A, B).

(A, B) must be in generalized Schur canonical form, that is, A and B are both upper triangular.

Syntax

Fortran specification:

```
use armpl_library

subroutine ctgsna(JOB, HOWMNY, SELECT, N, A, LDA, B, LDB, VL, LDVL, VR, LDVR,
                  S, DIF, MM, M, WORK, LWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ctgsna_(const char *job, const char *howmny, const armpl_int_t *select,
             const armpl_int_t *n, const armpl_singlecomplex_t *a,
             const armpl_int_t *lda, const armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, const armpl_singlecomplex_t *vl,
             const armpl_int_t *ldvl, const armpl_singlecomplex_t *vr,
             const armpl_int_t *ldvr, float *s, float *dif,
             const armpl_int_t *mm, armpl_int_t *m,
             armpl_singlecomplex_t *work, const armpl_int_t *lwork,
             armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

JOB Input parameter.

JOB is CHARACTER*1

Specifies whether condition numbers are required for eigenvalues (S) or eigenvectors (DIF): = 'E': for eigenvalues only (S); = 'V': for eigenvectors only (DIF); = 'B': for both eigenvalues and eigenvectors (S and DIF).

HOWMNY Input parameter.

HOWMNY is CHARACTER*1

= 'A': compute condition numbers for all eigenpairs; = 'S': compute condition numbers for selected eigenpairs specified by the array SELECT.

SELECT Input parameter.

SELECT is LOGICAL

SELECT is an array, dimension (N). If HOWMNY = 'S', SELECT specifies the eigenpairs for which condition numbers are required. To select condition numbers for the corresponding j-th eigenvalue and/or eigenvector, SELECT(j) must be set to .TRUE.. If HOWMNY = 'A', SELECT is not referenced.

N Input parameter.

N is INTEGER

The order of the square matrix pair (A, B). N >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The upper triangular matrix A in the pair (A, B).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, N). The upper triangular matrix B in the pair (A, B).

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

VL Input parameter.

VL is COMPLEX

VL is an array, dimension (LDVL, M). IF JOB = 'E' or 'B', VL must contain left eigenvectors of (A, B), corresponding to the eigenpairs specified by HOWMNY and SELECT. The eigenvectors must be stored in consecutive columns of VL, as returned by CTGEVC. If JOB = 'V', VL is not referenced.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the array VL. LDVL >= 1; and If JOB = 'E' or 'B', LDVL >= N.

VR Input parameter.

VR is COMPLEX

VR is an array, dimension (LDVR, M). IF JOB = 'E' or 'B', VR must contain right eigenvectors of (A, B), corresponding to the eigenpairs specified by HOWMNY and SELECT. The eigenvectors must be stored in consecutive columns of VR, as returned by CTGEVC. If JOB = 'V', VR is not referenced.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the array VR. LDVR >= 1; If JOB = 'E' or 'B', LDVR >= N.

S Output parameter.

S is REAL

S is an array, dimension (MM). If JOB = 'E' or 'B', the reciprocal condition numbers of the selected eigenvalues, stored in consecutive elements of the array. If JOB = 'V', S is not referenced.

DIF Output parameter.

DIF is REAL

DIF is an array, dimension (MM). If JOB = 'V' or 'B', the estimated reciprocal condition numbers of the selected eigenvectors, stored in consecutive elements of the array. If the eigenvalues cannot be reordered to compute DIF(j), DIF(j) is set to 0; this can only occur when the true value would be very small anyway. For each eigenvalue/vector specified by SELECT, DIF stores a Frobenius norm-based estimate of Difl. If JOB = 'E', DIF is not referenced.

MM Input parameter.

MM is INTEGER

The number of elements in the arrays S and DIF. MM >= M.

M Output parameter.

M is INTEGER

The number of elements of the arrays S and DIF used to store the specified condition numbers; for each selected eigenvalue one element is used. If HOWMNY = 'A', M is set to N.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N). If JOB = 'V' or 'B', LWORK >= max(1,2*N*N).

IWORK Output parameter.

IWORK is INTEGER array, dimension (N+2)

If JOB = 'E', IWORK is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit < 0: If INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dtgsna*, *stgsna* and *ztgsna*. It also exists with a native C interface as *LAPACKE_ctgsna*.

4.14.16 ctgsyl

ctgsyl solves the generalized Sylvester equation:

```
A * R - L * B = scale * C            (1)
D * R - L * E = scale * F
```

where R and L are unknown m-by-n matrices, (A, D), (B, E) and (C, F) are given matrix pairs of size m-by-m, n-by-n and m-by-n, respectively, with complex entries. A, B, D and E are upper triangular (i.e., (A,D) and (B,E) in generalized Schur form).

The solution (R, L) overwrites (C, F). 0 <= SCALE <= 1 is an output scaling factor chosen to avoid overflow.

In matrix notation (1) is equivalent to solve Zx = scale*b, where Z is defined as

```
Z = [ kron(In, A)  -kron(B**H, Im) ]         (2)
    [ kron(In, D)  -kron(E**H, Im) ],
```

Here Ix is the identity matrix of size x and X^H is the conjugate transpose of X. Kron(X, Y) is the Kronecker product between the matrices X and Y.

If TRANS = 'C', y in the conjugate transposed system Z^H *y = scale*b is solved for, which is equivalent to solve for R and L in

```
A**H * R + D**H * L = scale * C           (3)
R * B**H + L * E**H = scale * -F
```

This case (TRANS = 'C') is used to compute an one-norm-based estimate of Dif[(A,D), (B,E)], the separation between the matrix pairs (A,D) and (B,E), using CLACON.

If IJOB >= 1, `ctgsyl` computes a Frobenius norm-based estimate of Dif[(A,D),(B,E)]. That is, the reciprocal of a lower bound on the reciprocal of the smallest singular value of Z.

This is a level-3 BLAS algorithm.

Syntax

Fortran specification:

```
use armpl_library

subroutine ctgsyl(TRANS, IJOB, M, N, A, LDA, B, LDB, C, LDC, D, LDD, E, LDE,
                  F, LDF, SCALE, DIF, WORK, LWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ctgsyl_(const char *trans, const armpl_int_t *ijob, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_singlecomplex_t *a,
             const armpl_int_t *lda, const armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, armpl_singlecomplex_t *c,
             const armpl_int_t *ldc, const armpl_singlecomplex_t *d,
             const armpl_int_t *ldd, const armpl_singlecomplex_t *e,
             const armpl_int_t *lde, armpl_singlecomplex_t *f,
             const armpl_int_t *ldf, float *scale, float *dif,
             armpl_singlecomplex_t *work, const armpl_int_t *lwork,
             armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': solve the generalized sylvester equation (1). = 'C': solve the "conjugate transposed" system (3).

IJOB Input parameter.

IJOB is INTEGER

Specifies what kind of functionality to be performed. =0: solve (1) only. =1: The functionality of 0 and 3. =2: The functionality of 0 and 4. =3: Only an estimate of Dif[(A, D), (B, E)] is computed. (look ahead strategy is used). =4: Only an estimate of Dif[(A, D), (B, E)] is computed. (CGECON on sub-systems is used). Not referenced if TRANS = 'C'.

M Input parameter.

M is INTEGER

The order of the matrices A and D, and the row dimension of the matrices C, F, R and L.

N Input parameter.

N is INTEGER

The order of the matrices B and E, and the column dimension of the matrices C, F, R and L.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, M). The upper triangular matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, N). The upper triangular matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). On entry, C contains the right-hand-side of the first matrix equation in (1) or (3). On exit, if IJOB = 0, 1 or 2, C has been overwritten by the solution R. If IJOB = 3 or 4 and TRANS = 'N', C holds R, the solution achieved during the computation of the Dif-estimate.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

D Input parameter.

D is COMPLEX

D is an array, dimension (LDD, M). The upper triangular matrix D.

LDD Input parameter.

LDD is INTEGER

The leading dimension of the array D. LDD >= max(1, M).

E Input parameter.

E is COMPLEX

E is an array, dimension (LDE, N). The upper triangular matrix E.

LDE Input parameter.

LDE is INTEGER

The leading dimension of the array E. LDE >= max(1, N).

F Input and output parameter.

F is COMPLEX

F is an array, dimension (LDF, N). On entry, F contains the right-hand-side of the second matrix equation in (1) or (3). On exit, if IJOB = 0, 1 or 2, F has been overwritten by the solution L. If IJOB = 3 or 4 and TRANS = 'N', F holds L, the solution achieved during the computation of the Dif-estimate.

LDF Input parameter.

LDF is INTEGER

The leading dimension of the array F. LDF >= max(1, M).

DIF Output parameter.

DIF is REAL

On exit DIF is the reciprocal of a lower bound of the reciprocal of the Dif-function, i.e. DIF is an upper bound of Dif[(A, D), (B, E)] = sigma-min(Z), where Z as in (2). IF IJOB = 0 or TRANS = 'C', DIF is not referenced.

SCALE Output parameter.

SCALE is REAL

On exit SCALE is the scaling factor in (1) or (3). If 0 < SCALE < 1, C and F hold the solutions R and L, resp., to a slightly perturbed system but the input matrices A, B, D and E have not been changed. If SCALE = 0, R and L will hold the solutions to the homogenious system with C = F = 0.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK > = 1. If IJOB = 1 or 2 and TRANS = 'N', LWORK >= max(1,2*M*N).

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (M+N+2)

INFO Output parameter.

INFO is INTEGER

=0: successful exit <0: If INFO = -i, the i-th argument had an illegal value. >0: (A, D) and (B, E) have common or very close eigenvalues.

Related Information

For this routine in other precisions, please see *dtgsyl*, *stgsyl* and *ztgsyl*. It also exists with a native C interface as *LAPACKE_ctgsyl*.

4.14.17 dggbak

dggbak forms the right or left eigenvectors of a real generalized eigenvalue problem A*x = lambda*B*x, by backward transformation on the computed eigenvectors of the balanced pair of matrices output by DGGBAL.

Syntax

Fortran specification:

```
use armpl_library

subroutine dggbak(JOB, SIDE, N, ILO, IHI, LSCALE, RSCALE, M, V, LDV, INFO)
```

C specification:

```
#include "armpl.h"

void dggbak_(const char *job, const char *side, const armpl_int_t *n,
             const armpl_int_t *ilo, const armpl_int_t *ihi,
             const double *lscale, const double *rscale, const armpl_int_t *m,
             double *v, const armpl_int_t *ldv, armpl_int_t *info, ... );
```

Parameters

JOB Input parameter.

JOB is CHARACTER*1

Specifies the type of backward transformation required: = 'N': do nothing, return immediately; = 'P': do backward transformation for permutation only; = 'S': do backward transformation for scaling only; = 'B': do backward transformations for both permutation and scaling. JOB must be the same as the argument JOB supplied to DGGBAL.

SIDE Input parameter.

SIDE is CHARACTER*1

= 'R': V contains right eigenvectors; = 'L': V contains left eigenvectors.

N Input parameter.

N is INTEGER

The number of rows of the matrix V. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

The integers ILO and IHI determined by DGGBAL. 1 <= ILO <= IHI <= N, if N > 0; ILO=1 and IHI=0, if N=0.

LSCALE Input parameter.

LSCALE is DOUBLE PRECISION

LSCALE is an array, dimension (N). Details of the permutations and/or scaling factors applied to the left side of A and B, as returned by DGGBAL.

RSCALE Input parameter.

RSCALE is DOUBLE PRECISION

RSCALE is an array, dimension (N). Details of the permutations and/or scaling factors applied to the right side of A and B, as returned by DGGBAL.

M Input parameter.

M is INTEGER

The number of columns of the matrix V. M >= 0.

V Input and output parameter.

V is DOUBLE PRECISION

V is an array, dimension (LDV, M). On entry, the matrix of right or left eigenvectors to be transformed, as returned by DTGEVC. On exit, V is overwritten by the transformed eigenvectors.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the matrix V. LDV >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cggbak*, *sggbak* and *zggbak*. It also exists with a native C interface as *LAPACKE_dggbak*.

4.14.18 dggbal

dggbal balances a pair of general real matrices (A,B). This involves, first, permuting A and B by similarity transformations to isolate eigenvalues in the first 1 to ILO$-$1 and last IHI+1 to N elements on the diagonal; and second, applying a diagonal similarity transformation to rows and columns ILO to IHI to make the rows and columns as close in norm as possible. Both steps are optional.

Balancing may reduce the 1-norm of the matrices, and improve the accuracy of the computed eigenvalues and/or eigenvectors in the generalized eigenvalue problem A*x = lambda*B*x.

Syntax

Fortran specification:

```
use armpl_library

subroutine dggbal(JOB, N, A, LDA, B, LDB, ILO, IHI, LSCALE, RSCALE, WORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dggbal_(const char *job, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, double *b, const armpl_int_t *ldb,
             armpl_int_t *ilo, armpl_int_t *ihi, double *lscale,
             double *rscale, double *work, armpl_int_t *info, ... );
```

Parameters

JOB Input parameter.

JOB is CHARACTER*1

Specifies the operations to be performed on A and B: = 'N': none: simply set ILO = 1, IHI = N, LSCALE(I) = 1.0 and RSCALE(I) = 1.0 for i = 1,...,N. = 'P': permute only; = 'S': scale only; = 'B': both permute and scale.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the input matrix A. On exit, A is overwritten by the balanced matrix. If JOB = 'N', A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). On entry, the input matrix B. On exit, B is overwritten by the balanced matrix. If JOB = 'N', B is not referenced.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

ILO Output parameter.

ILO is INTEGER

IHI Output parameter.

IHI is INTEGER

ILO and IHI are set to integers such that on exit A(i,j) = 0 and B(i,j) = 0 if i > j and j = 1,. . . ,ILO-1 or i = IHI+1,. . . ,N. If JOB = 'N' or 'S', ILO = 1 and IHI = N.

LSCALE Output parameter.

LSCALE is DOUBLE PRECISION

LSCALE is an array, dimension (N). Details of the permutations and scaling factors applied to the left side of A and B. If P(j) is the index of the row interchanged with row j, and D(j) is the scaling factor applied to row j, then LSCALE(j) = P(j) for J = 1,. . . ,ILO-1 = D(j) for J = ILO,. . . ,IHI = P(j) for J = IHI+1,. . . ,N. The order in which the interchanges are made is N to IHI+1, then 1 to ILO-1.

RSCALE Output parameter.

RSCALE is DOUBLE PRECISION

RSCALE is an array, dimension (N). Details of the permutations and scaling factors applied to the right side of A and B. If P(j) is the index of the column interchanged with column j, and D(j) is the scaling factor applied to column j, then LSCALE(j) = P(j) for J = 1,. . . ,ILO-1 = D(j) for J = ILO,. . . ,IHI = P(j) for J = IHI+1,. . . ,N. The order in which the interchanges are made is N to IHI+1, then 1 to ILO-1.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (lwork). lwork must be at least max(1,6*N) when JOB = 'S' or 'B', and at least 1 when JOB = 'N' or 'P'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cggbal*, *sggbal* and *zggbal*. It also exists with a native C interface as *LAPACKE_dggbal*.

4.14.19 dgges

`dgges` computes for a pair of N-by-N real nonsymmetric matrices (A,B), the generalized eigenvalues, the generalized real Schur form (S,T), optionally, the left and/or right matrices of Schur vectors (VSL and VSR). This gives the generalized Schur factorization

```
(A,B) = ( (VSL)*S*(VSR)**T, (VSL)*T*(VSR)**T )
```

Optionally, it also orders the eigenvalues so that a selected cluster of eigenvalues appears in the leading diagonal blocks of the upper quasi-triangular matrix S and the upper triangular matrix T.The leading columns of VSL and VSR then form an orthonormal basis for the corresponding left and right eigenspaces (deflating subspaces).

(If only the generalized eigenvalues are needed, use the driver DGGEV instead, which is faster.)

A generalized eigenvalue for a pair of matrices (A,B) is a scalar w or a ratio alpha/beta = w, such that A - w*B is singular. It is usually represented as the pair (alpha,beta), as there is a reasonable interpretation for beta=0 or both being zero.

A pair of matrices (S,T) is in generalized real Schur form if T is upper triangular with non-negative diagonal and S is block upper triangular with 1-by-1 and 2-by-2 blocks. 1-by-1 blocks correspond to real generalized eigenvalues, while 2-by-2 blocks of S will be "standardized" by making the corresponding elements of T have the form:

```
[  a  0  ]
[  0  b  ]
```

and the pair of corresponding 2-by-2 blocks in S and T will have a complex conjugate pair of generalized eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgges(JOBVSL, JOBVSR, SORT, SELCTG, N, A, LDA, B, LDB, SDIM,
                 ALPHAR, ALPHAI, BETA, VSL, LDVSL, VSR, LDVSR, WORK, LWORK,
                 BWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgges_(const char *jobvsl, const char *jobvsr, const char *sort,
            ARMPL_DGGES_DELCTG delctg, const armpl_int_t *n, double *a,
            const armpl_int_t *lda, double *b, const armpl_int_t *ldb,
            armpl_int_t *sdim, double *alphar, double *alphai, double *beta,
            double *vsl, const armpl_int_t *ldvsl, double *vsr,
            const armpl_int_t *ldvsr, double *work, const armpl_int_t *lwork,
            armpl_int_t *bwork, armpl_int_t *info, ... );
```

Parameters

JOBVSL Input parameter.

JOBVSL is CHARACTER*1

= 'N': do not compute the left Schur vectors; = 'V': compute the left Schur vectors.

JOBVSR Input parameter.

JOBVSR is CHARACTER*1

= 'N': do not compute the right Schur vectors; = 'V': compute the right Schur vectors.

SORT Input parameter.

SORT is CHARACTER*1

Specifies whether or not to order the eigenvalues on the diagonal of the generalized Schur form. = 'N': Eigenvalues are not ordered; = 'S': Eigenvalues are ordered (see SELCTG);

SELCTG Input parameter.

SELCTG is a LOGICAL FUNCTION of three DOUBLE PRECISION arguments

SELCTG must be declared EXTERNAL in the calling subroutine. If SORT = 'N', SELCTG is not referenced. If SORT = 'S', SELCTG is used to select eigenvalues to sort to the top left of the Schur form. An eigenvalue (ALPHAR(j)+ALPHAI(j))/BETA(j) is selected if SELCTG(ALPHAR(j),ALPHAI(j),BETA(j)) is true; i.e. if either one of a complex conjugate pair of eigenvalues is selected, then both complex eigenvalues are selected.

Note that in the ill-conditioned case, a selected complex eigenvalue may no longer satisfy SELCTG(ALPHAR(j),ALPHAI(j), BETA(j)) = .TRUE. after ordering. INFO is to be set to N+2 in this case.

N Input parameter.

N is INTEGER

The order of the matrices A, B, VSL, and VSR. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the first of the pair of matrices. On exit, A has been overwritten by its generalized Schur form S.

LDA Input parameter.

LDA is INTEGER

The leading dimension of A. LDA >= max(1, N).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). On entry, the second of the pair of matrices. On exit, B has been overwritten by its generalized Schur form T.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B. LDB >= max(1, N).

SDIM Output parameter.

SDIM is INTEGER

If SORT = 'N', SDIM = 0. If SORT = 'S', SDIM = number of eigenvalues (after sorting) for which SELCTG is true. (Complex conjugate pairs for which SELCTG is true for either eigenvalue count as 2.)

ALPHAR Output parameter.

ALPHAR is DOUBLE PRECISION

ALPHAR is an array, dimension (N) .

ALPHAI Output parameter.

ALPHAI is DOUBLE PRECISION

ALPHAI is an array, dimension (N) .

BETA Output parameter.

BETA is DOUBLE PRECISION

BETA is an array, dimension (N). On exit, (ALPHAR(j) + ALPHAI(j)*i)/BETA(j), j=1,..., N, will be the generalized eigenvalues. ALPHAR(j) + ALPHAI(j)*i, and BETA(j),j=1,...,N are the diagonals of the complex Schur form (S,T) that would result if the 2-by-2 diagonal blocks of the real Schur form of (A, B) were further reduced to triangular form using 2-by-2 complex unitary transformations. If ALPHAI(j) is zero, then the j-th eigenvalue is real; if positive, then the j-th and (j+1)-st eigenvalues are a complex conjugate pair, with ALPHAI(j+1) negative.

Note: the quotients ALPHAR(j)/BETA(j) and ALPHAI(j)/BETA(j) may easily over- or underflow, and BETA(j) may even be zero. Thus, the user should avoid naively computing the ratio. However, ALPHAR and ALPHAI will be always less than and usually comparable with norm(A) in magnitude, and BETA always less than and usually comparable with norm(B).

VSL Output parameter.

VSL is DOUBLE PRECISION

VSL is an array, dimension (LDVSL, N). If JOBVSL = 'V', VSL will contain the left Schur vectors. Not referenced if JOBVSL = 'N'.

LDVSL Input parameter.

LDVSL is INTEGER

The leading dimension of the matrix VSL. LDVSL >=1, and if JOBVSL = 'V', LDVSL >= N.

VSR Output parameter.

VSR is DOUBLE PRECISION

VSR is an array, dimension (LDVSR, N). If JOBVSR = 'V', VSR will contain the right Schur vectors. Not referenced if JOBVSR = 'N'.

LDVSR Input parameter.

LDVSR is INTEGER

The leading dimension of the matrix VSR. LDVSR >= 1, and if JOBVSR = 'V', LDVSR >= N.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If N = 0, LWORK >= 1, else LWORK >= 8*N+16. For good performance , LWORK must generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

BWORK Output parameter.

BWORK is LOGICAL

BWORK is an array, dimension (N). Not referenced if SORT = 'N'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. = 1,...,N: The QZ iteration failed. (A, B) are not in Schur form, but ALPHAR(j), ALPHAI(j), and BETA(j) should be correct for j=INFO+1,...,N. > N: =N+1: other than QZ iteration failed in DHGEQZ. =N+2: after reordering, roundoff changed values of some complex eigenvalues so that leading eigenvalues in the Generalized Schur form no longer satisfy SELCTG=.TRUE. This could also be caused due to scaling. =N+3: reordering failed in DTGSEN.

Related Information

For this routine in other precisions, please see *cgges*, *sgges* and *zgges*. It also exists with a native C interface as *LAPACKE_dgges*.

4.14.20 dgges3

dgges3 computes for a pair of N-by-N real nonsymmetric matrices (A,B), the generalized eigenvalues, the generalized real Schur form (S,T), optionally, the left and/or right matrices of Schur vectors (VSL and VSR). This gives the generalized Schur factorization

```
(A,B) = ( (VSL)*S*(VSR)**T, (VSL)*T*(VSR)**T )
```

Optionally, it also orders the eigenvalues so that a selected cluster of eigenvalues appears in the leading diagonal blocks of the upper quasi-triangular matrix S and the upper triangular matrix T.The leading columns of VSL and VSR then form an orthonormal basis for the corresponding left and right eigenspaces (deflating subspaces).

(If only the generalized eigenvalues are needed, use the driver DGGEV instead, which is faster.)

A generalized eigenvalue for a pair of matrices (A,B) is a scalar w or a ratio alpha/beta = w, such that A - w*B is singular. It is usually represented as the pair (alpha,beta), as there is a reasonable interpretation for beta=0 or both being zero.

A pair of matrices (S,T) is in generalized real Schur form if T is upper triangular with non-negative diagonal and S is block upper triangular with 1-by-1 and 2-by-2 blocks. 1-by-1 blocks correspond to real generalized eigenvalues, while 2-by-2 blocks of S will be "standardized" by making the corresponding elements of T have the form:

```
[  a  0  ]
[  0  b  ]
```

and the pair of corresponding 2-by-2 blocks in S and T will have a complex conjugate pair of generalized eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgges3(JOBVSL, JOBVSR, SORT, SELCTG, N, A, LDA, B, LDB, SDIM,
                  ALPHAR, ALPHAI, BETA, VSL, LDVSL, VSR, LDVSR, WORK, LWORK,
                  BWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgges3_(const char *jobvsl, const char *jobvsr, const char *sort,
             ARMPL_DGGES3_SELCTG selctg, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, double *b, const armpl_int_t *ldb,
             armpl_int_t *sdim, double *alphar, double *alphai, double *beta,
             double *vsl, const armpl_int_t *ldvsl, double *vsr,
```

(continues on next page)

(continued from previous page)

```
            const armpl_int_t *ldvsr, double *work, const armpl_int_t *lwork,
            armpl_int_t *bwork, armpl_int_t *info, ... );
```

Parameters

JOBVSL Input parameter.

JOBVSL is CHARACTER*1

= 'N': do not compute the left Schur vectors; = 'V': compute the left Schur vectors.

JOBVSR Input parameter.

JOBVSR is CHARACTER*1

= 'N': do not compute the right Schur vectors; = 'V': compute the right Schur vectors.

SORT Input parameter.

SORT is CHARACTER*1

Specifies whether or not to order the eigenvalues on the diagonal of the generalized Schur form. = 'N': Eigenvalues are not ordered; = 'S': Eigenvalues are ordered (see SELCTG);

SELCTG Input parameter.

SELCTG is a LOGICAL FUNCTION of three DOUBLE PRECISION arguments

SELCTG must be declared EXTERNAL in the calling subroutine. If SORT = 'N', SELCTG is not referenced. If SORT = 'S', SELCTG is used to select eigenvalues to sort to the top left of the Schur form. An eigenvalue (ALPHAR(j)+ALPHAI(j))/BETA(j) is selected if SELCTG(ALPHAR(j),ALPHAI(j),BETA(j)) is true; i.e. if either one of a complex conjugate pair of eigenvalues is selected, then both complex eigenvalues are selected.

Note that in the ill-conditioned case, a selected complex eigenvalue may no longer satisfy SELCTG(ALPHAR(j),ALPHAI(j), BETA(j)) = .TRUE. after ordering. INFO is to be set to N+2 in this case.

N Input parameter.

N is INTEGER

The order of the matrices A, B, VSL, and VSR. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the first of the pair of matrices. On exit, A has been overwritten by its generalized Schur form S.

LDA Input parameter.

LDA is INTEGER

The leading dimension of A. LDA >= max(1, N).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). On entry, the second of the pair of matrices. On exit, B has been overwritten by its generalized Schur form T.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B. LDB >= max(1, N).

SDIM Output parameter.

SDIM is INTEGER

If SORT = 'N', SDIM = 0. If SORT = 'S', SDIM = number of eigenvalues (after sorting) for which SELCTG is true. (Complex conjugate pairs for which SELCTG is true for either eigenvalue count as 2.)

ALPHAR Output parameter.

ALPHAR is DOUBLE PRECISION

ALPHAR is an array, dimension (N) .

ALPHAI Output parameter.

ALPHAI is DOUBLE PRECISION

ALPHAI is an array, dimension (N) .

BETA Output parameter.

BETA is DOUBLE PRECISION

BETA is an array, dimension (N). On exit, (ALPHAR(j) + ALPHAI(j)*i)/BETA(j), j=1,..., N, will be the generalized eigenvalues. ALPHAR(j) + ALPHAI(j)*i, and BETA(j),j=1,...,N are the diagonals of the complex Schur form (S,T) that would result if the 2-by-2 diagonal blocks of the real Schur form of (A, B) were further reduced to triangular form using 2-by-2 complex unitary transformations. If ALPHAI(j) is zero, then the j-th eigenvalue is real; if positive, then the j-th and (j+1)-st eigenvalues are a complex conjugate pair, with ALPHAI(j+1) negative.

Note: the quotients ALPHAR(j)/BETA(j) and ALPHAI(j)/BETA(j) may easily over- or underflow, and BETA(j) may even be zero. Thus, the user should avoid naively computing the ratio. However, ALPHAR and ALPHAI will be always less than and usually comparable with norm(A) in magnitude, and BETA always less than and usually comparable with norm(B).

VSL Output parameter.

VSL is DOUBLE PRECISION

VSL is an array, dimension (LDVSL, N). If JOBVSL = 'V', VSL will contain the left Schur vectors. Not referenced if JOBVSL = 'N'.

LDVSL Input parameter.

LDVSL is INTEGER

The leading dimension of the matrix VSL. LDVSL >=1, and if JOBVSL = 'V', LDVSL >= N.

VSR Output parameter.

VSR is DOUBLE PRECISION

VSR is an array, dimension (LDVSR, N). If JOBVSR = 'V', VSR will contain the right Schur vectors. Not referenced if JOBVSR = 'N'.

LDVSR Input parameter.

LDVSR is INTEGER

The leading dimension of the matrix VSR. LDVSR >= 1, and if JOBVSR = 'V', LDVSR >= N.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

BWORK Output parameter.

BWORK is LOGICAL

BWORK is an array, dimension (N). Not referenced if SORT = 'N'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. = 1,...,N: The QZ iteration failed. (A, B) are not in Schur form, but ALPHAR(j), ALPHAI(j), and BETA(j) should be correct for j=INFO+1,...,N. > N: =N+1: other than QZ iteration failed in DHGEQZ. =N+2: after reordering, roundoff changed values of some complex eigenvalues so that leading eigenvalues in the Generalized Schur form no longer satisfy SELCTG=.TRUE. This could also be caused due to scaling. =N+3: reordering failed in DTGSEN.

Related Information

For this routine in other precisions, please see *cgges3*, *sgges3* and *zgges3*. It also exists with a native C interface as *LAPACKE_dgges3*.

4.14.21 dggesx

dggesx computes for a pair of N-by-N real nonsymmetric matrices (A,B), the generalized eigenvalues, the real Schur form (S,T), and, optionally, the left and/or right matrices of Schur vectors (VSL and VSR). This gives the generalized Schur factorization

```
(A,B) = ( (VSL) S (VSR)**T, (VSL) T (VSR)**T )
```

Optionally, it also orders the eigenvalues so that a selected cluster of eigenvalues appears in the leading diagonal blocks of the upper quasi-triangular matrix S and the upper triangular matrix T; computes a reciprocal condition number for the average of the selected eigenvalues (RCONDE); and computes a reciprocal condition number for the right and left deflating subspaces corresponding to the selected eigenvalues (RCONDV). The leading columns of VSL and VSR then form an orthonormal basis for the corresponding left and right eigenspaces (deflating subspaces).

A generalized eigenvalue for a pair of matrices (A,B) is a scalar w or a ratio alpha/beta = w, such that A - w*B is singular. It is usually represented as the pair (alpha,beta), as there is a reasonable interpretation for beta=0 or for both being zero.

A pair of matrices (S,T) is in generalized real Schur form if T is upper triangular with non-negative diagonal and S is block upper triangular with 1-by-1 and 2-by-2 blocks. 1-by-1 blocks correspond to real generalized eigenvalues, while 2-by-2 blocks of S will be "standardized" by making the corresponding elements of T have the form:

```
[  a  0  ]
[  0  b  ]
```

and the pair of corresponding 2-by-2 blocks in S and T will have a complex conjugate pair of generalized eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine dggesx(JOBVSL, JOBVSR, SORT, SELCTG, SENSE, N, A, LDA, B, LDB,
                  SDIM, ALPHAR, ALPHAI, BETA, VSL, LDVSL, VSR, LDVSR, RCONDE,
                  RCONDV, WORK, LWORK, IWORK, LIWORK, BWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dggesx_(const char *jobvsl, const char *jobvsr, const char *sort,
             ARMPL_DGGESX_DELCTG delctg, const char *sense,
             const armpl_int_t *n, double *a, const armpl_int_t *lda,
             double *b, const armpl_int_t *ldb, armpl_int_t *sdim,
             double *alphar, double *alphai, double *beta, double *vsl,
             const armpl_int_t *ldvsl, double *vsr, const armpl_int_t *ldvsr,
             double *rconde, double *rcondv, double *work,
             const armpl_int_t *lwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *bwork, armpl_int_t *info,
             ... );
```

Parameters

JOBVSL Input parameter.

JOBVSL is CHARACTER*1

= 'N': do not compute the left Schur vectors; = 'V': compute the left Schur vectors.

JOBVSR Input parameter.

JOBVSR is CHARACTER*1

= 'N': do not compute the right Schur vectors; = 'V': compute the right Schur vectors.

SORT Input parameter.

SORT is CHARACTER*1

Specifies whether or not to order the eigenvalues on the diagonal of the generalized Schur form. = 'N': Eigenvalues are not ordered; = 'S': Eigenvalues are ordered (see SELCTG).

SELCTG Input parameter.

SELCTG is a LOGICAL FUNCTION of three DOUBLE PRECISION arguments

SELCTG must be declared EXTERNAL in the calling subroutine. If SORT = 'N', SELCTG is not referenced. If SORT = 'S', SELCTG is used to select eigenvalues to sort to the top left of the Schur form. An eigenvalue (ALPHAR(j)+ALPHAI(j))/BETA(j) is selected if SELCTG(ALPHAR(j),ALPHAI(j),BETA(j)) is true; i.e. if either one of a complex conjugate pair of eigenvalues is selected, then both complex eigenvalues are selected. Note that a selected complex eigenvalue may no longer satisfy SELCTG(ALPHAR(j),ALPHAI(j),BETA(j)) = .TRUE. after ordering, since ordering may change the value of complex eigenvalues (especially if the eigenvalue is ill-conditioned), in this case INFO is set to N+3.

SENSE Input parameter.

SENSE is CHARACTER*1

Determines which reciprocal condition numbers are computed. = 'N' : None are computed; = 'E' : Computed for average of selected eigenvalues only; = 'V' : Computed for selected deflating subspaces only; = 'B' : Computed for both. If SENSE = 'E', 'V', or 'B', SORT must equal 'S'.

N Input parameter.

N is INTEGER

The order of the matrices A, B, VSL, and VSR. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the first of the pair of matrices. On exit, A has been overwritten by its generalized Schur form S.

LDA Input parameter.

LDA is INTEGER

The leading dimension of A. LDA >= max(1, N).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). On entry, the second of the pair of matrices. On exit, B has been overwritten by its generalized Schur form T.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B. LDB >= max(1, N).

SDIM Output parameter.

SDIM is INTEGER

If SORT = 'N', SDIM = 0. If SORT = 'S', SDIM = number of eigenvalues (after sorting) for which SELCTG is true. (Complex conjugate pairs for which SELCTG is true for either eigenvalue count as 2.)

ALPHAR Output parameter.

ALPHAR is DOUBLE PRECISION

ALPHAR is an array, dimension (N) .

ALPHAI Output parameter.

ALPHAI is DOUBLE PRECISION

ALPHAI is an array, dimension (N) .

BETA Output parameter.

BETA is DOUBLE PRECISION

BETA is an array, dimension (N). On exit, (ALPHAR(j) + ALPHAI(j)*i)/BETA(j), j=1,..., N, will be the generalized eigenvalues. ALPHAR(j) + ALPHAI(j)*i and BETA(j),j=1,...,N are the diagonals of the complex Schur form (S,T) that would result if the 2-by-2 diagonal blocks of the real Schur form of (A, B) were further reduced to triangular form using 2-by-2 complex unitary transformations. If ALPHAI(j) is zero, then the j-th eigenvalue is real; if positive, then the j-th and (j+1)-st eigenvalues are a complex conjugate pair, with ALPHAI(j+1) negative.

Note: the quotients ALPHAR(j)/BETA(j) and ALPHAI(j)/BETA(j) may easily over- or underflow, and BETA(j) may even be zero. Thus, the user should avoid naively computing the ratio. However, ALPHAR and ALPHAI will be always less than and usually comparable with norm(A) in magnitude, and BETA always less than and usually comparable with norm(B).

VSL Output parameter.

VSL is DOUBLE PRECISION

VSL is an array, dimension (LDVSL, N). If JOBVSL = 'V', VSL will contain the left Schur vectors. Not referenced if JOBVSL = 'N'.

LDVSL Input parameter.

LDVSL is INTEGER

The leading dimension of the matrix VSL. LDVSL >=1, and if JOBVSL = 'V', LDVSL >= N.

VSR Output parameter.

VSR is DOUBLE PRECISION

VSR is an array, dimension (LDVSR, N). If JOBVSR = 'V', VSR will contain the right Schur vectors. Not referenced if JOBVSR = 'N'.

LDVSR Input parameter.

LDVSR is INTEGER

The leading dimension of the matrix VSR. LDVSR >= 1, and if JOBVSR = 'V', LDVSR >= N.

RCONDE Output parameter.

RCONDE is DOUBLE PRECISION

RCONDE is an array, dimension (2). If SENSE = 'E' or 'B', RCONDE(1) and RCONDE(2) contain the reciprocal condition numbers for the average of the selected eigenvalues. Not referenced if SENSE = 'N' or 'V'.

RCONDV Output parameter.

RCONDV is DOUBLE PRECISION

RCONDV is an array, dimension (2). If SENSE = 'V' or 'B', RCONDV(1) and RCONDV(2) contain the reciprocal condition numbers for the selected deflating subspaces. Not referenced if SENSE = 'N' or 'E'.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If N = 0, LWORK >= 1, else if SENSE = 'E', 'V', or 'B', LWORK >= max(8*N, 6*N+16, 2*SDIM*(N-SDIM)), else LWORK >= max(8*N, 6*N+16). Note that 2*SDIM*(N-SDIM) <= N*N/2. Note also that an error is only returned if LWORK < max(8*N, 6*N+16), but if SENSE = 'E' or 'V' or 'B' this may not be large enough.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the bound on the optimal size of the WORK array and the minimum size of the IWORK array, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the minimum LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. If SENSE = 'N' or N = 0, LIWORK >= 1, otherwise LIWORK >= N+6.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the bound on the optimal size of the WORK array and the minimum size of the IWORK array, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

BWORK Output parameter.

BWORK is LOGICAL

BWORK is an array, dimension (N). Not referenced if SORT = 'N'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. = 1,...,N: The QZ iteration failed. (A, B) are not in Schur form, but ALPHAR(j), ALPHAI(j), and BETA(j) should be correct for j=INFO+1,...,N. > N: =N+1: other than QZ iteration failed in DHGEQZ =N+2: after reordering, roundoff changed values of some complex eigenvalues so that leading eigenvalues in the Generalized Schur form no longer satisfy SELCTG=.TRUE. This could also be caused due to scaling. =N+3: reordering failed in DTGSEN.

Related Information

For this routine in other precisions, please see *cggesx*, *sggesx* and *zggesx*. It also exists with a native C interface as *LAPACKE_dggesx*.

4.14.22 dggev

dggev computes for a pair of N-by-N real nonsymmetric matrices (A,B) the generalized eigenvalues, and optionally, the left and/or right generalized eigenvectors.

A generalized eigenvalue for a pair of matrices (A,B) is a scalar lambda or a ratio alpha/beta = lambda, such that A - lambda*B is singular. It is usually represented as the pair (alpha,beta), as there is a reasonable interpretation for beta=0, and even for both being zero.

The right eigenvector v(j) corresponding to the eigenvalue lambda(j) of (A,B) satisfies

```
A * v(j) = lambda(j) * B * v(j).
```

The left eigenvector u(j) corresponding to the eigenvalue lambda(j) of (A,B) satisfies

```
u(j)**H * A  = lambda(j) * u(j)**H * B .
```

where $u(j)^H$ is the conjugate-transpose of u(j).

Syntax

Fortran specification:

```
use armpl_library

subroutine dggev(JOBVL, JOBVR, N, A, LDA, B, LDB, ALPHAR, ALPHAI, BETA, VL,
                 LDVL, VR, LDVR, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dggev_(const char *jobvl, const char *jobvr, const armpl_int_t *n,
            double *a, const armpl_int_t *lda, double *b,
            const armpl_int_t *ldb, double *alphar, double *alphai,
            double *beta, double *vl, const armpl_int_t *ldvl, double *vr,
            const armpl_int_t *ldvr, double *work, const armpl_int_t *lwork,
            armpl_int_t *info, ... );
```

Parameters

JOBVL Input parameter.

JOBVL is CHARACTER*1

= 'N': do not compute the left generalized eigenvectors; = 'V': compute the left generalized eigenvectors.

JOBVR Input parameter.

JOBVR is CHARACTER*1

= 'N': do not compute the right generalized eigenvectors; = 'V': compute the right generalized eigenvectors.

N Input parameter.

N is INTEGER

The order of the matrices A, B, VL, and VR. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the matrix A in the pair (A, B). On exit, A has been overwritten.

LDA Input parameter.

LDA is INTEGER

The leading dimension of A. LDA >= max(1, N).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). On entry, the matrix B in the pair (A, B). On exit, B has been overwritten.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B. LDB >= max(1, N).

ALPHAR Output parameter.

ALPHAR is DOUBLE PRECISION

ALPHAR is an array, dimension (N) .

ALPHAI Output parameter.

ALPHAI is DOUBLE PRECISION

ALPHAI is an array, dimension (N) .

BETA Output parameter.

BETA is DOUBLE PRECISION

BETA is an array, dimension (N). On exit, (ALPHAR(j) + ALPHAI(j)*i)/BETA(j), j=1,..., N, will be the generalized eigenvalues. If ALPHAI(j) is zero, then the j-th eigenvalue is real; if positive, then the j-th and (j+1)-st eigenvalues are a complex conjugate pair, with ALPHAI(j+1) negative.

Note: the quotients ALPHAR(j)/BETA(j) and ALPHAI(j)/BETA(j) may easily over- or underflow, and BETA(j) may even be zero. Thus, the user should avoid naively computing the ratio alpha/beta. However, ALPHAR and ALPHAI will be always less than and usually comparable with norm(A) in magnitude, and BETA always less than and usually comparable with norm(B).

VL Output parameter.

VL is DOUBLE PRECISION

VL is an array, dimension (LDVL, N). If JOBVL = 'V', the left eigenvectors u(j) are stored one after another in the columns of VL, in the same order as their eigenvalues. If the j-th eigenvalue is real, then u(j) = VL(:,j), the j-th column of VL. If the j-th and (j+1)-th eigenvalues form a complex conjugate pair, then u(j) = VL(:,j)+i*VL(:,j+1) and u(j+1) = VL(:,j)-i*VL(:,j+1). Each eigenvector is scaled so the largest component has abs(real part)+abs(imag. part)=1. Not referenced if JOBVL = 'N'.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the matrix VL. LDVL >= 1, and if JOBVL = 'V', LDVL >= N.

VR Output parameter.

VR is DOUBLE PRECISION

VR is an array, dimension (LDVR, N). If JOBVR = 'V', the right eigenvectors v(j) are stored one after another in the columns of VR, in the same order as their eigenvalues. If the j-th eigenvalue is real, then v(j) = VR(:,j), the j-th column of VR. If the j-th and (j+1)-th eigenvalues form a complex conjugate pair, then v(j) = VR(:,j)+i*VR(:,j+1) and v(j+1) = VR(:,j)-i*VR(:,j+1). Each eigenvector is scaled so the largest component has abs(real part)+abs(imag. part)=1. Not referenced if JOBVR = 'N'.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the matrix VR. LDVR >= 1, and if JOBVR = 'V', LDVR >= N.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,8*N). For good performance, LWORK must generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. = 1,...,N: The QZ iteration failed. No eigenvectors have been calculated, but ALPHAR(j), ALPHAI(j), and BETA(j) should be correct for j=INFO+1,...,N. > N: =N+1: other than QZ iteration failed in DHGEQZ. =N+2: error return from DTGEVC.

Related Information

For this routine in other precisions, please see *cggev*, *sggev* and *zggev*. It also exists with a native C interface as *LAPACKE_dggev*.

4.14.23 dggev3

dggev3 computes for a pair of N-by-N real nonsymmetric matrices (A,B) the generalized eigenvalues, and optionally, the left and/or right generalized eigenvectors.

A generalized eigenvalue for a pair of matrices (A,B) is a scalar lambda or a ratio alpha/beta = lambda, such that A - lambda*B is singular. It is usually represented as the pair (alpha,beta), as there is a reasonable interpretation for beta=0, and even for both being zero.

The right eigenvector v(j) corresponding to the eigenvalue lambda(j) of (A,B) satisfies

```
A * v(j) = lambda(j) * B * v(j).
```

The left eigenvector u(j) corresponding to the eigenvalue lambda(j) of (A,B) satisfies

```
u(j)**H * A  = lambda(j) * u(j)**H * B .
```

where $u(j)^H$ is the conjugate-transpose of u(j).

Syntax

Fortran specification:

```
use armpl_library

subroutine dggev3(JOBVL, JOBVR, N, A, LDA, B, LDB, ALPHAR, ALPHAI, BETA, VL,
                  LDVL, VR, LDVR, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dggev3_(const char *jobvl, const char *jobvr, const armpl_int_t *n,
             double *a, const armpl_int_t *lda, double *b,
             const armpl_int_t *ldb, double *alphar, double *alphai,
             double *beta, double *vl, const armpl_int_t *ldvl, double *vr,
             const armpl_int_t *ldvr, double *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

JOBVL Input parameter.

JOBVL is CHARACTER*1

= 'N': do not compute the left generalized eigenvectors; = 'V': compute the left generalized eigenvectors.

JOBVR Input parameter.

JOBVR is CHARACTER*1

= 'N': do not compute the right generalized eigenvectors; = 'V': compute the right generalized eigenvectors.

N Input parameter.

N is INTEGER

The order of the matrices A, B, VL, and VR. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the matrix A in the pair (A, B). On exit, A has been overwritten.

LDA Input parameter.

LDA is INTEGER

The leading dimension of A. LDA >= max(1, N).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). On entry, the matrix B in the pair (A, B). On exit, B has been overwritten.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B. LDB >= max(1, N).

ALPHAR Output parameter.

ALPHAR is DOUBLE PRECISION

ALPHAR is an array, dimension (N) .

ALPHAI Output parameter.

ALPHAI is DOUBLE PRECISION

ALPHAI is an array, dimension (N) .

BETA Output parameter.

BETA is DOUBLE PRECISION

BETA is an array, dimension (N). On exit, (ALPHAR(j) + ALPHAI(j)*i)/BETA(j), j=1,..., N, will be the generalized eigenvalues. If ALPHAI(j) is zero, then the j-th eigenvalue is real; if positive, then the j-th and (j+1)-st eigenvalues are a complex conjugate pair, with ALPHAI(j+1) negative.

Note: the quotients ALPHAR(j)/BETA(j) and ALPHAI(j)/BETA(j) may easily over- or underflow, and BETA(j) may even be zero. Thus, the user should avoid naively computing the ratio alpha/beta. However, ALPHAR and ALPHAI will be always less than and usually comparable with norm(A) in magnitude, and BETA always less than and usually comparable with norm(B).

VL Output parameter.

VL is DOUBLE PRECISION

VL is an array, dimension (LDVL, N). If JOBVL = 'V', the left eigenvectors u(j) are stored one after another in the columns of VL, in the same order as their eigenvalues. If the j-th eigenvalue is real, then u(j) = VL(:,j), the j-th column of VL. If the j-th and (j+1)-th eigenvalues form a complex conjugate pair, then u(j) = VL(:,j)+i*VL(:,j+1) and u(j+1) = VL(:,j)-i*VL(:,j+1). Each eigenvector is scaled so the largest component has abs(real part)+abs(imag. part)=1. Not referenced if JOBVL = 'N'.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the matrix VL. LDVL >= 1, and if JOBVL = 'V', LDVL >= N.

VR Output parameter.

VR is DOUBLE PRECISION

VR is an array, dimension (LDVR, N). If JOBVR = 'V', the right eigenvectors v(j) are stored one after another in the columns of VR, in the same order as their eigenvalues. If the j-th eigenvalue is real, then v(j) = VR(:,j), the j-th column of VR. If the j-th and (j+1)-th eigenvalues form a complex conjugate pair, then v(j) = VR(:,j)+i*VR(:,j+1) and v(j+1) = VR(:,j)-i*VR(:,j+1). Each eigenvector is scaled so the largest component has abs(real part)+abs(imag. part)=1. Not referenced if JOBVR = 'N'.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the matrix VR. LDVR >= 1, and if JOBVR = 'V', LDVR >= N.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. = 1,...,N: The QZ iteration failed. No eigenvectors have been calculated, but ALPHAR(j), ALPHAI(j), and BETA(j) should be correct for j=INFO+1,...,N. > N: =N+1: other than QZ iteration failed in DHGEQZ. =N+2: error return from DTGEVC.

Related Information

For this routine in other precisions, please see *cggev3*, *sggev3* and *zggev3*. It also exists with a native C interface as *LAPACKE_dggev3*.

4.14.24 dggevx

dggevx computes for a pair of N-by-N real nonsymmetric matrices (A,B) the generalized eigenvalues, and optionally, the left and/or right generalized eigenvectors.

Optionally also, it computes a balancing transformation to improve the conditioning of the eigenvalues and eigenvectors (ILO, IHI, LSCALE, RSCALE, ABNRM, and BBNRM), reciprocal condition numbers for the eigenvalues (RCONDE), and reciprocal condition numbers for the right eigenvectors (RCONDV).

A generalized eigenvalue for a pair of matrices (A,B) is a scalar lambda or a ratio alpha/beta = lambda, such that A - lambda*B is singular. It is usually represented as the pair (alpha,beta), as there is a reasonable interpretation for beta=0, and even for both being zero.

The right eigenvector v(j) corresponding to the eigenvalue lambda(j) of (A,B) satisfies

```
A * v(j) = lambda(j) * B * v(j) .
```

The left eigenvector u(j) corresponding to the eigenvalue lambda(j) of (A,B) satisfies

```
u(j)**H * A  = lambda(j) * u(j)**H * B.
```

where $u(j)^H$ is the conjugate-transpose of u(j).

Syntax

Fortran specification:

```
use armpl_library

subroutine dggevx(BALANC, JOBVL, JOBVR, SENSE, N, A, LDA, B, LDB, ALPHAR,
                  ALPHAI, BETA, VL, LDVL, VR, LDVR, ILO, IHI, LSCALE, RSCALE,
                  ABNRM, BBNRM, RCONDE, RCONDV, WORK, LWORK, IWORK, BWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dggevx_(const char *balanc, const char *jobvl, const char *jobvr,
             const char *sense, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, double *b, const armpl_int_t *ldb,
             double *alphar, double *alphai, double *beta, double *vl,
             const armpl_int_t *ldvl, double *vr, const armpl_int_t *ldvr,
             armpl_int_t *ilo, armpl_int_t *ihi, double *lscale,
             double *rscale, double *abnrm, double *bbnrm, double *rconde,
             double *rcondv, double *work, const armpl_int_t *lwork,
             armpl_int_t *iwork, armpl_int_t *bwork, armpl_int_t *info,
             ... );
```

Parameters

BALANC Input parameter.

BALANC is CHARACTER*1

Specifies the balance option to be performed. = 'N': do not diagonally scale or permute; = 'P': permute only; = 'S': scale only; = 'B': both permute and scale. Computed reciprocal condition numbers will be for the matrices after permuting and/or balancing. Permuting does not change condition numbers (in exact arithmetic), but balancing does.

JOBVL Input parameter.

JOBVL is CHARACTER*1

= 'N': do not compute the left generalized eigenvectors; = 'V': compute the left generalized eigenvectors.

JOBVR Input parameter.

JOBVR is CHARACTER*1

= 'N': do not compute the right generalized eigenvectors; = 'V': compute the right generalized eigenvectors.

SENSE Input parameter.

SENSE is CHARACTER*1

Determines which reciprocal condition numbers are computed. = 'N': none are computed; = 'E': computed for eigenvalues only; = 'V': computed for eigenvectors only; = 'B': computed for eigenvalues and eigenvectors.

N Input parameter.

N is INTEGER

The order of the matrices A, B, VL, and VR. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the matrix A in the pair (A, B). On exit, A has been overwritten. If JOBVL='V' or JOBVR='V' or both, then A contains the first part of the real Schur form of the "balanced" versions of the input A and B.

LDA Input parameter.

LDA is INTEGER

The leading dimension of A. LDA >= max(1, N).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). On entry, the matrix B in the pair (A, B). On exit, B has been overwritten. If JOBVL='V' or JOBVR='V' or both, then B contains the second part of the real Schur form of the "balanced" versions of the input A and B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B. LDB >= max(1, N).

ALPHAR Output parameter.

ALPHAR is DOUBLE PRECISION

ALPHAR is an array, dimension (N) .

ALPHAI Output parameter.

ALPHAI is DOUBLE PRECISION

ALPHAI is an array, dimension (N) .

BETA Output parameter.

BETA is DOUBLE PRECISION

BETA is an array, dimension (N). On exit, (ALPHAR(j) + ALPHAI(j)*i)/BETA(j), j=1,..., N, will be the generalized eigenvalues. If ALPHAI(j) is zero, then the j-th eigenvalue is real; if positive, then the j-th and (j+1)-st eigenvalues are a complex conjugate pair, with ALPHAI(j+1) negative.

Note: the quotients ALPHAR(j)/BETA(j) and ALPHAI(j)/BETA(j) may easily over- or underflow, and BETA(j) may even be zero. Thus, the user should avoid naively computing the ratio ALPHA/BETA. However, ALPHAR and ALPHAI will be always less than and usually comparable with norm(A) in magnitude, and BETA always less than and usually comparable with norm(B).

VL Output parameter.

VL is DOUBLE PRECISION

VL is an array, dimension (LDVL, N). If JOBVL = 'V', the left eigenvectors u(j) are stored one after another in the columns of VL, in the same order as their eigenvalues. If the j-th eigenvalue is real, then u(j) = VL(:,j), the j-th column of VL. If the j-th and (j+1)-th eigenvalues form a complex conjugate pair, then u(j) = VL(:,j)+i*VL(:,j+1) and u(j+1) = VL(:,j)-i*VL(:,j+1). Each eigenvector will be scaled so the largest component have abs(real part) + abs(imag. part) = 1. Not referenced if JOBVL = 'N'.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the matrix VL. LDVL >= 1, and if JOBVL = 'V', LDVL >= N.

VR Output parameter.

VR is DOUBLE PRECISION

VR is an array, dimension (LDVR, N). If JOBVR = 'V', the right eigenvectors v(j) are stored one after another in the columns of VR, in the same order as their eigenvalues. If the j-th eigenvalue is real, then v(j) = VR(:,j), the j-th column of VR. If the j-th and (j+1)-th eigenvalues form a complex conjugate pair, then v(j) = VR(:,j)+i*VR(:,j+1) and v(j+1) = VR(:,j)-i*VR(:,j+1). Each eigenvector will be scaled so the largest component have abs(real part) + abs(imag. part) = 1. Not referenced if JOBVR = 'N'.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the matrix VR. LDVR >= 1, and if JOBVR = 'V', LDVR >= N.

ILO Output parameter.

ILO is INTEGER

IHI Output parameter.

IHI is INTEGER

ILO and IHI are integer values such that on exit A(i,j) = 0 and B(i,j) = 0 if i > j and j = 1,...,ILO-1 or i = IHI+1,...,N. If BALANC = 'N' or 'S', ILO = 1 and IHI = N.

LSCALE Output parameter.

LSCALE is DOUBLE PRECISION

LSCALE is an array, dimension (N). Details of the permutations and scaling factors applied to the left side of A and B. If PL(j) is the index of the row interchanged with row j, and DL(j) is the scaling factor applied to row j, then LSCALE(j) = PL(j) for j = 1,...,ILO-1 = DL(j) for j = ILO,...,IHI = PL(j) for j = IHI+1,...,N. The order in which the interchanges are made is N to IHI+1, then 1 to ILO-1.

RSCALE Output parameter.

RSCALE is DOUBLE PRECISION

RSCALE is an array, dimension (N). Details of the permutations and scaling factors applied to the right side of A and B. If PR(j) is the index of the column interchanged with column j, and DR(j) is the scaling factor applied to column j, then RSCALE(j) = PR(j) for j = 1,...,ILO-1 = DR(j) for j = ILO,...,IHI = PR(j) for j = IHI+1,...,N The order in which the interchanges are made is N to IHI+1, then 1 to ILO-1.

ABNRM Output parameter.

ABNRM is DOUBLE PRECISION

The one-norm of the balanced matrix A.

BBNRM Output parameter.

BBNRM is DOUBLE PRECISION

The one-norm of the balanced matrix B.

RCONDE Output parameter.

RCONDE is DOUBLE PRECISION

RCONDE is an array, dimension (N). If SENSE = 'E' or 'B', the reciprocal condition numbers of the eigenvalues, stored in consecutive elements of the array. For a complex conjugate pair of eigenvalues two consecutive elements of RCONDE are set to the same value. Thus RCONDE(j), RCONDV(j), and the j-th columns of VL and VR all correspond to the j-th eigenpair. If SENSE = 'N or 'V', RCONDE is not referenced.

RCONDV Output parameter.

RCONDV is DOUBLE PRECISION

RCONDV is an array, dimension (N). If SENSE = 'V' or 'B', the estimated reciprocal condition numbers of the eigenvectors, stored in consecutive elements of the array. For a complex eigenvector two consecutive elements of RCONDV are set to the same value. If the eigenvalues cannot be reordered to compute RCONDV(j), RCONDV(j) is set to 0; this can only occur when the true value would be very small anyway. If SENSE = 'N' or 'E', RCONDV is not referenced.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,2*N). If BALANC = 'S' or 'B', or JOBVL = 'V', or JOBVR = 'V', LWORK >= max(1,6*N). If SENSE = 'E' or 'B', LWORK >= max(1,10*N). If SENSE = 'V' or 'B', LWORK >= 2*N*N+8*N+16.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (N+6)

If SENSE = 'E', IWORK is not referenced.

BWORK Output parameter.

BWORK is LOGICAL

BWORK is an array, dimension (N). If SENSE = 'N', BWORK is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. = 1,...,N: The QZ iteration failed. No eigenvectors have been calculated, but ALPHAR(j), ALPHAI(j), and BETA(j) should be correct for j=INFO+1,...,N. > N: =N+1: other than QZ iteration failed in DHGEQZ. =N+2: error return from DTGEVC.

Related Information

For this routine in other precisions, please see *cggevx*, *sggevx* and *zggevx*. It also exists with a native C interface as *LAPACKE_dggevx*.

4.14.25 dgghd3

dgghd3 reduces a pair of real matrices (A,B) to generalized upper Hessenberg form using orthogonal transformations, where A is a general matrix and B is upper triangular. The form of the generalized eigenvalue problem is

```
A*x = lambda*B*x,
```

and B is typically made upper triangular by computing its QR factorization and moving the orthogonal matrix Q to the left side of the equation.

This subroutine simultaneously reduces A to a Hessenberg matrix H:

```
Q**T*A*Z = H
```

and transforms B to another upper triangular matrix T:

```
Q**T*B*Z = T
```

in order to reduce the problem to its standard form

```
H*y = lambda*T*y
```

where y = Z^T *x.

The orthogonal matrices Q and Z are determined as products of Givens rotations. They may either be formed explicitly, or they may be postmultiplied into input matrices Q1 and Z1, so that

```
 Q1 * A * Z1**T = (Q1*Q) * H * (Z1*Z)**T

Q1 * B * Z1**T = (Q1*Q) * T * (Z1*Z)**T
```

If Q1 is the orthogonal matrix from the QR factorization of B in the original equation A*x = lambda*B*x, then `dgghd3` reduces the original problem to generalized Hessenberg form.

This is a blocked variant of DGGHRD, using matrix-matrix multiplications for parts of the computation to enhance performance.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgghd3(COMPQ, COMPZ, N, ILO, IHI, A, LDA, B, LDB, Q, LDQ, Z, LDZ,
                  WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgghd3_(const char *compq, const char *compz, const armpl_int_t *n,
             const armpl_int_t *ilo, const armpl_int_t *ihi, double *a,
             const armpl_int_t *lda, double *b, const armpl_int_t *ldb,
             double *q, const armpl_int_t *ldq, double *z,
             const armpl_int_t *ldz, double *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

COMPQ Input parameter.

COMPQ is CHARACTER*1

= 'N': do not compute Q; = 'I': Q is initialized to the unit matrix, and the orthogonal matrix Q is returned; = 'V': Q must contain an orthogonal matrix Q1 on entry, and the product Q1* Q is returned.

COMPZ Input parameter.

COMPZ is CHARACTER*1

= 'N': do not compute Z; = 'I': Z is initialized to the unit matrix, and the orthogonal matrix Z is returned; = 'V': Z must contain an orthogonal matrix Z1 on entry, and the product Z1* Z is returned.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

ILO and IHI mark the rows and columns of A which are to be reduced. It is assumed that A is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N. ILO and IHI are normally set by a previous call to DGGBAL; otherwise they should be set to 1 and N respectively. 1 <= ILO <= IHI <= N, if N > 0; ILO=1 and IHI=0, if N=0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the N-by-N general matrix to be reduced. On exit, the upper triangle and the first subdiagonal of A are overwritten with the upper Hessenberg matrix H, and the rest is set to zero.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). On entry, the N-by-N upper triangular matrix B. On exit, the upper triangular matrix T = Q^T B Z. The elements below the diagonal are set to zero.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Q Input and output parameter.

Q is DOUBLE PRECISION

Q is an array, dimension (LDQ, N). On entry, if COMPQ = 'V', the orthogonal matrix Q1, typically from the QR factorization of B. On exit, if COMPQ='I', the orthogonal matrix Q, and if COMPQ = 'V', the product Q1*Q. Not referenced if COMPQ='N'.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= N if COMPQ='V' or 'I'; LDQ >= 1 otherwise.

Z Input and output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, N). On entry, if COMPZ = 'V', the orthogonal matrix Z1. On exit, if COMPZ='I', the orthogonal matrix Z, and if COMPZ = 'V', the product Z1*Z. Not referenced if COMPZ='N'.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= N if COMPZ='V' or 'I'; LDZ >= 1 otherwise.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (LWORK). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= 1. For optimum performance LWORK >= 6*N*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cgghd3*, *sgghd3* and *zgghd3*. It also exists with a native C interface as *LAPACKE_dgghd3*.

4.14.26 dgghrd

dgghrd reduces a pair of real matrices (A,B) to generalized upper Hessenberg form using orthogonal transformations, where A is a general matrix and B is upper triangular. The form of the generalized eigenvalue problem is

```
A*x = lambda*B*x,
```

and B is typically made upper triangular by computing its QR factorization and moving the orthogonal matrix Q to the left side of the equation.

This subroutine simultaneously reduces A to a Hessenberg matrix H:

```
Q**T*A*Z = H
```

and transforms B to another upper triangular matrix T:

```
Q**T*B*Z = T
```

in order to reduce the problem to its standard form

```
H*y = lambda*T*y
```

where y = Z^T *x.

The orthogonal matrices Q and Z are determined as products of Givens rotations. They may either be formed explicitly, or they may be postmultiplied into input matrices Q1 and Z1, so that

```
 Q1 * A * Z1**T = (Q1*Q) * H * (Z1*Z)**T

Q1 * B * Z1**T = (Q1*Q) * T * (Z1*Z)**T
```

If Q1 is the orthogonal matrix from the QR factorization of B in the original equation A*x = lambda*B*x, then dgghrd reduces the original problem to generalized Hessenberg form.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgghrd(COMPQ, COMPZ, N, ILO, IHI, A, LDA, B, LDB, Q, LDQ, Z, LDZ,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dgghrd_(const char *compq, const char *compz, const armpl_int_t *n,
             const armpl_int_t *ilo, const armpl_int_t *ihi, double *a,
             const armpl_int_t *lda, double *b, const armpl_int_t *ldb,
             double *q, const armpl_int_t *ldq, double *z,
             const armpl_int_t *ldz, armpl_int_t *info, ... );
```

Parameters

COMPQ Input parameter.

COMPQ is CHARACTER*1

= 'N': do not compute Q; = 'I': Q is initialized to the unit matrix, and the orthogonal matrix Q is returned; = 'V': Q must contain an orthogonal matrix Q1 on entry, and the product Q1* Q is returned.

COMPZ Input parameter.

COMPZ is CHARACTER*1

= 'N': do not compute Z; = 'I': Z is initialized to the unit matrix, and the orthogonal matrix Z is returned; = 'V': Z must contain an orthogonal matrix Z1 on entry, and the product Z1* Z is returned.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

ILO and IHI mark the rows and columns of A which are to be reduced. It is assumed that A is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N. ILO and IHI are normally set by a previous call to DGGBAL; otherwise they should be set to 1 and N respectively. 1 <= ILO <= IHI <= N, if N > 0; ILO=1 and IHI=0, if N=0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the N-by-N general matrix to be reduced. On exit, the upper triangle and the first subdiagonal of A are overwritten with the upper Hessenberg matrix H, and the rest is set to zero.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). On entry, the N-by-N upper triangular matrix B. On exit, the upper triangular matrix T = Q^T B Z. The elements below the diagonal are set to zero.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Q Input and output parameter.

Q is DOUBLE PRECISION

Q is an array, dimension (LDQ, N). On entry, if COMPQ = 'V', the orthogonal matrix Q1, typically from the QR factorization of B. On exit, if COMPQ='I', the orthogonal matrix Q, and if COMPQ = 'V', the product Q1*Q. Not referenced if COMPQ='N'.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= N if COMPQ='V' or 'I'; LDQ >= 1 otherwise.

Z Input and output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, N). On entry, if COMPZ = 'V', the orthogonal matrix Z1. On exit, if COMPZ='I', the orthogonal matrix Z, and if COMPZ = 'V', the product Z1*Z. Not referenced if COMPZ='N'.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= N if COMPZ='V' or 'I'; LDZ >= 1 otherwise.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cgghrd*, *sgghrd* and *zgghrd*. It also exists with a native C interface as *LAPACKE_dgghrd*.

4.14.27 dhgeqz

dhgeqz computes the eigenvalues of a real matrix pair (H,T), where H is an upper Hessenberg matrix and T is upper triangular, using the double-shift QZ method. Matrix pairs of this type are produced by the reduction to generalized upper Hessenberg form of a real matrix pair (A,B):

```
A = Q1*H*Z1**T,  B = Q1*T*Z1**T,
```

as computed by DGGHRD.

If JOB='S', then the Hessenberg-triangular pair (H,T) is also reduced to generalized Schur form,

```
H = Q*S*Z**T,  T = Q*P*Z**T,
```

where Q and Z are orthogonal matrices, P is an upper triangular matrix, and S is a quasi-triangular matrix with 1-by-1 and 2-by-2 diagonal blocks.

The 1-by-1 blocks correspond to real eigenvalues of the matrix pair (H,T) and the 2-by-2 blocks correspond to complex conjugate pairs of eigenvalues.

Additionally, the 2-by-2 upper triangular diagonal blocks of P corresponding to 2-by-2 blocks of S are reduced to positive diagonal form, i.e., if S(j+1,j) is non-zero, then P(j+1,j) = P(j,j+1) = 0, P(j,j) > 0, and P(j+1,j+1) > 0.

Optionally, the orthogonal matrix Q from the generalized Schur factorization may be postmultiplied into an input matrix Q1, and the orthogonal matrix Z may be postmultiplied into an input matrix Z1. If Q1 and Z1 are the orthogonal matrices from DGGHRD that reduced the matrix pair (A,B) to generalized upper Hessenberg form, then the output matrices Q1*Q and Z1*Z are the orthogonal factors from the generalized Schur factorization of (A,B):

```
A = (Q1*Q)*S*(Z1*Z)**T,  B = (Q1*Q)*P*(Z1*Z)**T.
```

To avoid overflow, eigenvalues of the matrix pair (H,T) (equivalently, of (A,B)) are computed as a pair of values (alpha,beta), where alpha is complex and beta real. If beta is nonzero, lambda = alpha / beta is an eigenvalue of the generalized nonsymmetric eigenvalue problem (GNEP)

```
A*x = lambda*B*x
```

and if alpha is nonzero, mu = beta / alpha is an eigenvalue of the alternate form of the GNEP

```
mu*A*y = B*y.
```

Real eigenvalues can be read directly from the generalized Schur form:

```
alpha = S(i,i), beta = P(i,i).
```

Ref: C.B. Moler & G.W. Stewart, “An Algorithm for Generalized Matrix Eigenvalue Problems”, SIAM J. Numer. Anal., 10(1973), pp. 241–256.

Syntax

Fortran specification:

```
use armpl_library

subroutine dhgeqz(JOB, COMPQ, COMPZ, N, ILO, IHI, H, LDH, T, LDT, ALPHAR,
                  ALPHAI, BETA, Q, LDQ, Z, LDZ, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dhgeqz_(const char *job, const char *compq, const char *compz,
             const armpl_int_t *n, const armpl_int_t *ilo,
             const armpl_int_t *ihi, double *h, const armpl_int_t *ldh,
             double *t, const armpl_int_t *ldt, double *alphar,
             double *alphai, double *beta, double *q, const armpl_int_t *ldq,
             double *z, const armpl_int_t *ldz, double *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

JOB Input parameter.

JOB is CHARACTER*1

= ‘E’: Compute eigenvalues only; = ‘S’: Compute eigenvalues and the Schur form.

COMPQ Input parameter.

COMPQ is CHARACTER*1

= ‘N’: Left Schur vectors (Q) are not computed; = ‘I’: Q is initialized to the unit matrix and the matrix Q of left Schur vectors of (H, T) is returned; = ‘V’: Q must contain an orthogonal matrix Q1 on entry and the product Q1* Q is returned.

COMPZ Input parameter.

COMPZ is CHARACTER*1

= ‘N’: Right Schur vectors (Z) are not computed; = ‘I’: Z is initialized to the unit matrix and the matrix Z of right Schur vectors of (H, T) is returned; = ‘V’: Z must contain an orthogonal matrix Z1 on entry and the product Z1* Z is returned.

N Input parameter.

N is INTEGER

The order of the matrices H, T, Q, and Z. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

ILO and IHI mark the rows and columns of H which are in Hessenberg form. It is assumed that A is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N. If N > 0, 1 <= ILO <= IHI <= N; if N = 0, ILO=1 and IHI=0.

H Input and output parameter.

H is DOUBLE PRECISION

H is an array, dimension (LDH, N). On entry, the N-by-N upper Hessenberg matrix H. On exit, if JOB = 'S', H contains the upper quasi-triangular matrix S from the generalized Schur factorization. If JOB = 'E', the diagonal blocks of H match those of S, but the rest of H is unspecified.

LDH Input parameter.

LDH is INTEGER

The leading dimension of the array H. LDH >= max(1, N).

T Input and output parameter.

T is DOUBLE PRECISION

T is an array, dimension (LDT, N). On entry, the N-by-N upper triangular matrix T. On exit, if JOB = 'S', T contains the upper triangular matrix P from the generalized Schur factorization; 2-by-2 diagonal blocks of P corresponding to 2-by-2 blocks of S are reduced to positive diagonal form, i.e., if H(j+1,j) is non-zero, then T(j+1,j) = T(j,j+1) = 0, T(j,j) > 0, and T(j+1,j+1) > 0. If JOB = 'E', the diagonal blocks of T match those of P, but the rest of T is unspecified.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

ALPHAR Output parameter.

ALPHAR is DOUBLE PRECISION

ALPHAR is an array, dimension (N). The real parts of each scalar alpha defining an eigenvalue of GNEP.

ALPHAI Output parameter.

ALPHAI is DOUBLE PRECISION

ALPHAI is an array, dimension (N). The imaginary parts of each scalar alpha defining an eigenvalue of GNEP. If ALPHAI(j) is zero, then the j-th eigenvalue is real; if positive, then the j-th and (j+1)-st eigenvalues are a complex conjugate pair, with ALPHAI(j+1) = -ALPHAI(j).

BETA Output parameter.

BETA is DOUBLE PRECISION

BETA is an array, dimension (N). The scalars beta that define the eigenvalues of GNEP. Together, the quantities alpha = (ALPHAR(j),ALPHAI(j)) and beta = BETA(j) represent the j-th eigenvalue of the matrix pair (A,B), in one of the forms lambda = alpha/beta or mu = beta/alpha. Since either lambda or mu may overflow, they should not, in general, be computed.

Q Input and output parameter.

Q is DOUBLE PRECISION

Q is an array, dimension (LDQ, N). On entry, if COMPQ = 'V', the orthogonal matrix Q1 used in the reduction of (A,B) to generalized Hessenberg form. On exit, if COMPQ = 'I', the orthogonal matrix of left Schur vectors

of (H, T), and if COMPQ = 'V', the orthogonal matrix of left Schur vectors of (A,B). Not referenced if COMPQ = 'N'.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= 1. If COMPQ='V' or 'I', then LDQ >= N.

Z Input and output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, N). On entry, if COMPZ = 'V', the orthogonal matrix Z1 used in the reduction of (A,B) to generalized Hessenberg form. On exit, if COMPZ = 'I', the orthogonal matrix of right Schur vectors of (H, T), and if COMPZ = 'V', the orthogonal matrix of right Schur vectors of (A,B). Not referenced if COMPZ = 'N'.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1. If COMPZ='V' or 'I', then LDZ >= N.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO >= 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N).

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value = 1,...,N: the QZ iteration did not converge. (H, T) is not in Schur form, but ALPHAR(i), ALPHAI(i), and BETA(i), i=INFO+1,...,N should be correct. = N+1,...,2*N: the shift calculation failed. (H, T) is not in Schur form, but ALPHAR(i), ALPHAI(i), and BETA(i), i=INFO-N+1,...,N should be correct.

Related Information

For this routine in other precisions, please see *chgeqz*, *shgeqz* and *zhgeqz*. It also exists with a native C interface as *LAPACKE_dhgeqz*.

4.14.28 dtgevc

dtgevc computes some or all of the right and/or left eigenvectors of a pair of real matrices (S,P), where S is a quasi-triangular matrix and P is upper triangular. Matrix pairs of this type are produced by the generalized Schur factorization of a matrix pair (A,B):

```
A = Q*S*Z**T,  B = Q*P*Z**T
```

as computed by DGGHRD + DHGEQZ.

The right eigenvector x and the left eigenvector y of (S,P) corresponding to an eigenvalue w are defined by:

```
S*x = w*P*x,   (y**H)*S = w*(y**H)*P,
```

where y^H denotes the conjugate tranpose of y. The eigenvalues are not input to this routine, but are computed directly from the diagonal blocks of S and P.

This routine returns the matrices X and/or Y of right and left eigenvectors of (S,P), or the products Z*X and/or Q*Y, where Z and Q are input matrices. If Q and Z are the orthogonal factors from the generalized Schur factorization of a matrix pair (A,B), then Z*X and Q*Y are the matrices of right and left eigenvectors of (A,B).

Syntax

Fortran specification:

```
use armpl_library

subroutine dtgevc(SIDE, HOWMNY, SELECT, N, S, LDS, P, LDP, VL, LDVL, VR, LDVR,
                  MM, M, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dtgevc_(const char *side, const char *howmny, const armpl_int_t *select,
             const armpl_int_t *n, const double *s, const armpl_int_t *lds,
             const double *p, const armpl_int_t *ldp, double *vl,
             const armpl_int_t *ldvl, double *vr, const armpl_int_t *ldvr,
             const armpl_int_t *mm, armpl_int_t *m, double *work,
             armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'R': compute right eigenvectors only; = 'L': compute left eigenvectors only; = 'B': compute both right and left eigenvectors.

HOWMNY Input parameter.

HOWMNY is CHARACTER*1

= 'A': compute all right and/or left eigenvectors; = 'B': compute all right and/or left eigenvectors, backtransformed by the matrices in VR and/or VL; = 'S': compute selected right and/or left eigenvectors, specified by the logical array SELECT.

SELECT Input parameter.

SELECT is LOGICAL

SELECT is an array, dimension (N). If HOWMNY='S', SELECT specifies the eigenvectors to be computed. If w(j) is a real eigenvalue, the corresponding real eigenvector is computed if SELECT(j) is .TRUE.. If w(j) and w(j+1) are the real and imaginary parts of a complex eigenvalue, the corresponding complex eigenvector is computed if either SELECT(j) or SELECT(j+1) is .TRUE., and on exit SELECT(j) is set to .TRUE. and SELECT(j+1) is set to .FALSE.. Not referenced if HOWMNY = 'A' or 'B'.

N Input parameter.

N is INTEGER

The order of the matrices S and P. N >= 0.

S Input parameter.

S is DOUBLE PRECISION

S is an array, dimension (LDS, N). The upper quasi-triangular matrix S from a generalized Schur factorization, as computed by DHGEQZ.

LDS Input parameter.

LDS is INTEGER

The leading dimension of array S. LDS >= max(1, N).

P Input parameter.

P is DOUBLE PRECISION

P is an array, dimension (LDP, N). The upper triangular matrix P from a generalized Schur factorization, as computed by DHGEQZ. 2-by-2 diagonal blocks of P corresponding to 2-by-2 blocks of S must be in positive diagonal form.

LDP Input parameter.

LDP is INTEGER

The leading dimension of array P. LDP >= max(1, N).

VL Input and output parameter.

VL is DOUBLE PRECISION

VL is an array, dimension (LDVL, MM). On entry, if SIDE = 'L' or 'B' and HOWMNY = 'B', VL must contain an N-by-N matrix Q (usually the orthogonal matrix Q of left Schur vectors returned by DHGEQZ). On exit, if SIDE = 'L' or 'B', VL contains: if HOWMNY = 'A', the matrix Y of left eigenvectors of (S, P); if HOWMNY = 'B', the matrix Q*Y; if HOWMNY = 'S', the left eigenvectors of (S, P) specified by SELECT, stored consecutively in the columns of VL, in the same order as their eigenvalues.

A complex eigenvector corresponding to a complex eigenvalue is stored in two consecutive columns, the first holding the real part, and the second the imaginary part.

Not referenced if SIDE = 'R'.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of array VL. LDVL >= 1, and if SIDE = 'L' or 'B', LDVL >= N.

VR Input and output parameter.

VR is DOUBLE PRECISION

VR is an array, dimension (LDVR, MM). On entry, if SIDE = 'R' or 'B' and HOWMNY = 'B', VR must contain an N-by-N matrix Z (usually the orthogonal matrix Z of right Schur vectors returned by DHGEQZ).

On exit, if SIDE = 'R' or 'B', VR contains: if HOWMNY = 'A', the matrix X of right eigenvectors of (S, P); if HOWMNY = 'B' or 'b', the matrix Z*X; if HOWMNY = 'S' or 's', the right eigenvectors of (S, P) specified by SELECT, stored consecutively in the columns of VR, in the same order as their eigenvalues.

A complex eigenvector corresponding to a complex eigenvalue is stored in two consecutive columns, the first holding the real part and the second the imaginary part.

Not referenced if SIDE = 'L'.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the array VR. LDVR >= 1, and if SIDE = 'R' or 'B', LDVR >= N.

MM Input parameter.

MM is INTEGER

The number of columns in the arrays VL and/or VR. MM >= M.

M Output parameter.

M is INTEGER

The number of columns in the arrays VL and/or VR actually used to store the eigenvectors. If HOWMNY = 'A' or 'B', M is set to N. Each selected real eigenvector occupies one column and each selected complex eigenvector occupies two columns.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (6*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: the 2-by-2 block (INFO:INFO+1) does not have a complex eigenvalue.

Related Information

For this routine in other precisions, please see *ctgevc*, *stgevc* and *ztgevc*. It also exists with a native C interface as *LAPACKE_dtgevc*.

4.14.29 dtgexc

dtgexc reorders the generalized real Schur decomposition of a real matrix pair (A,B) using an orthogonal equivalence transformation

```
(A, B) = Q * (A, B) * Z**T,
```

so that the diagonal block of (A, B) with row index IFST is moved to row ILST.

(A, B) must be in generalized real Schur canonical form (as returned by DGGES), i.e. A is block upper triangular with 1-by-1 and 2-by-2 diagonal blocks. B is upper triangular.

Optionally, the matrices Q and Z of generalized Schur vectors are updated.

```
Q(in) * A(in) * Z(in)**T = Q(out) * A(out) * Z(out)**T
Q(in) * B(in) * Z(in)**T = Q(out) * B(out) * Z(out)**T
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dtgexc(WANTQ, WANTZ, N, A, LDA, B, LDB, Q, LDQ, Z, LDZ, IFST, ILST,
                  WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dtgexc_(const armpl_int_t *wantq, const armpl_int_t *wantz,
             const armpl_int_t *n, double *a, const armpl_int_t *lda,
             double *b, const armpl_int_t *ldb, double *q,
             const armpl_int_t *ldq, double *z, const armpl_int_t *ldz,
             armpl_int_t *ifst, armpl_int_t *ilst, double *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

WANTQ Input parameter.

WANTQ is LOGICAL

.TRUE. : update the left transformation matrix Q; .FALSE.: do not update Q.

WANTZ Input parameter.

WANTZ is LOGICAL

.TRUE. : update the right transformation matrix Z; .FALSE.: do not update Z.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the matrix A in generalized real Schur canonical form. On exit, the updated matrix A, again in generalized real Schur canonical form.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). On entry, the matrix B in generalized real Schur canonical form (A, B). On exit, the updated matrix B, again in generalized real Schur canonical form (A, B).

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Q Input and output parameter.

Q is DOUBLE PRECISION

Q is an array, dimension (LDQ, N). On entry, if WANTQ = .TRUE., the orthogonal matrix Q. On exit, the updated matrix Q. If WANTQ = .FALSE., Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= 1. If WANTQ = .TRUE., LDQ >= N.

Z Input and output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, N). On entry, if WANTZ = .TRUE., the orthogonal matrix Z. On exit, the updated matrix Z. If WANTZ = .FALSE., Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1. If WANTZ = .TRUE., LDZ >= N.

IFST Input and output parameter.

IFST is INTEGER

ILST Input and output parameter.

ILST is INTEGER

Specify the reordering of the diagonal blocks of (A, B). The block with row index IFST is moved to row ILST, by a sequence of swapping between adjacent blocks. On exit, if IFST pointed on entry to the second row of a 2-by-2 block, it is changed to point to the first row; ILST always points to the first row of the block in its final position (which may differ from its input value by +1 or -1). 1 <= IFST, ILST <= N.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= 1 when N <= 1, otherwise LWORK >= 4* N + 16.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

=0: successful exit. <0: if INFO = -i, the i-th argument had an illegal value. =1: The transformed matrix pair (A, B) would be too far from generalized Schur form; the problem is ill- conditioned. (A, B) may have been partially reordered, and ILST points to the first row of the current position of the block being moved.

Related Information

For this routine in other precisions, please see *ctgexc*, *stgexc* and *ztgexc*. It also exists with a native C interface as *LAPACKE_dtgexc*.

4.14.30 dtgsen

dtgsen reorders the generalized real Schur decomposition of a real matrix pair (A, B) (in terms of an orthonormal equivalence trans- formation Q^T * (A, B) * Z), so that a selected cluster of eigenvalues appears in the leading diagonal blocks of the upper quasi-triangular matrix A and the upper triangular B. The leading columns of Q and Z form orthonormal bases of the corresponding left and right eigen- spaces (deflating subspaces). (A, B) must be in generalized real Schur canonical form (as returned by DGGES), i.e. A is block upper triangular with 1-by-1 and 2-by-2 diagonal blocks. B is upper triangular.

dtgsen also computes the generalized eigenvalues

```
w(j) = (ALPHAR(j) + i*ALPHAI(j))/BETA(j)
```

of the reordered matrix pair (A, B).

Optionally, dtgsen computes the estimates of reciprocal condition numbers for eigenvalues and eigenspaces. These are Difu[(A11,B11), (A22,B22)] and Difl[(A11,B11), (A22,B22)], i.e. the separation(s) between the matrix pairs (A11, B11) and (A22,B22) that correspond to the selected cluster and the eigenvalues outside the cluster, resp., and norms of "projections" onto left and right eigenspaces w.r.t. the selected cluster in the (1,1)-block.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtgsen(IJOB, WANTQ, WANTZ, SELECT, N, A, LDA, B, LDB, ALPHAR,
                  ALPHAI, BETA, Q, LDQ, Z, LDZ, M, PL, PR, DIF, WORK, LWORK,
                  IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dtgsen_(const armpl_int_t *ijob, const armpl_int_t *wantq,
             const armpl_int_t *wantz, const armpl_int_t *select,
             const armpl_int_t *n, double *a, const armpl_int_t *lda,
             double *b, const armpl_int_t *ldb, double *alphar,
             double *alphai, double *beta, double *q, const armpl_int_t *ldq,
             double *z, const armpl_int_t *ldz, armpl_int_t *m, double *pl,
             double *pr, double *dif, double *work, const armpl_int_t *lwork,
             armpl_int_t *iwork, const armpl_int_t *liwork,
             armpl_int_t *info);
```

Parameters

IJOB Input parameter.

IJOB is INTEGER

Specifies whether condition numbers are required for the cluster of eigenvalues (PL and PR) or the deflating subspaces (Difu and Difl): =0: Only reorder w.r.t. SELECT. No extras. =1: Reciprocal of norms of "projections" onto left and right eigenspaces w.r.t. the selected cluster (PL and PR). =2: Upper bounds on Difu and Difl. F-norm-based estimate (DIF(1:2)). =3: Estimate of Difu and Difl. 1-norm-based estimate (DIF(1:2)). About 5 times as expensive as IJOB = 2. =4: Compute PL, PR and DIF (i.e. 0, 1 and 2 above): Economic version to get it all. =5: Compute PL, PR and DIF (i.e. 0, 1 and 3 above)

WANTQ Input parameter.

WANTQ is LOGICAL

.TRUE. : update the left transformation matrix Q; .FALSE.: do not update Q.

WANTZ Input parameter.

WANTZ is LOGICAL

.TRUE. : update the right transformation matrix Z; .FALSE.: do not update Z.

SELECT Input parameter.

SELECT is LOGICAL

SELECT is an array, dimension (N). SELECT specifies the eigenvalues in the selected cluster. To select a real eigenvalue w(j), SELECT(j) must be set to .TRUE.. To select a complex conjugate pair of eigenvalues

w(j) and w(j+1), corresponding to a 2-by-2 diagonal block, either SELECT(j) or SELECT(j+1) or both must be set to .TRUE.; a complex conjugate pair of eigenvalues must be either both included in the cluster or both excluded.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension(LDA, N). On entry, the upper quasi-triangular matrix A, with (A, B) in generalized real Schur canonical form. On exit, A is overwritten by the reordered matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension(LDB, N). On entry, the upper triangular matrix B, with (A, B) in generalized real Schur canonical form. On exit, B is overwritten by the reordered matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

ALPHAR Output parameter.

ALPHAR is DOUBLE PRECISION

ALPHAR is an array, dimension (N) .

ALPHAI Output parameter.

ALPHAI is DOUBLE PRECISION

ALPHAI is an array, dimension (N) .

BETA Output parameter.

BETA is DOUBLE PRECISION

BETA is an array, dimension (N). On exit, (ALPHAR(j) + ALPHAI(j)*i)/BETA(j), j=1,..., N, will be the generalized eigenvalues. ALPHAR(j) + ALPHAI(j)*i and BETA(j),j=1,...,N are the diagonals of the complex Schur form (S,T) that would result if the 2-by-2 diagonal blocks of the real generalized Schur form of (A, B) were further reduced to triangular form using complex unitary transformations. If ALPHAI(j) is zero, then the j-th eigenvalue is real; if positive, then the j-th and (j+1)-st eigenvalues are a complex conjugate pair, with ALPHAI(j+1) negative.

Q Input and output parameter.

Q is DOUBLE PRECISION

Q is an array, dimension (LDQ, N). On entry, if WANTQ = .TRUE., Q is an N-by-N matrix. On exit, Q has been postmultiplied by the left orthogonal transformation matrix which reorder (A, B); The leading M columns of Q form orthonormal bases for the specified pair of left eigenspaces (deflating subspaces). If WANTQ = .FALSE., Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= 1; and if WANTQ = .TRUE., LDQ >= N.

Z Input and output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, N). On entry, if WANTZ = .TRUE., Z is an N-by-N matrix. On exit, Z has been postmultiplied by the left orthogonal transformation matrix which reorder (A, B); The leading M columns of Z form orthonormal bases for the specified pair of left eigenspaces (deflating subspaces). If WANTZ = .FALSE., Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1; If WANTZ = .TRUE., LDZ >= N.

M Output parameter.

M is INTEGER

The dimension of the specified pair of left and right eigen- spaces (deflating subspaces). 0 <= M <= N.

PL Output parameter.

PL is DOUBLE PRECISION

PR Output parameter.

PR is DOUBLE PRECISION

If IJOB = 1, 4 or 5, PL, PR are lower bounds on the reciprocal of the norm of "projections" onto left and right eigenspaces with respect to the selected cluster. 0 < PL, PR <= 1. If M = 0 or M = N, PL = PR = 1. If IJOB = 0, 2 or 3, PL and PR are not referenced.

DIF Output parameter.

DIF is DOUBLE PRECISION

DIF is an array, dimension (2).. If IJOB >= 2, DIF(1:2) store the estimates of Difu and Difl. If IJOB = 2 or 4, DIF(1:2) are F-norm-based upper bounds on Difu and Difl. If IJOB = 3 or 5, DIF(1:2) are 1-norm-based estimates of Difu and Difl. If M = 0 or N, DIF(1:2) = F-norm([A, B]). If IJOB = 0 or 1, DIF is not referenced.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= 4*N+16. If IJOB = 1, 2 or 4, LWORK >= MAX(4*N+16, 2*M*(N-M)). If IJOB = 3 or 5, LWORK >= MAX(4*N+16, 4*M*(N-M)).

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. LIWORK >= 1. If IJOB = 1, 2 or 4, LIWORK >= N+6. If IJOB = 3 or 5, LIWORK >= MAX(2*M*(N-M), N+6).

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the IWORK array, returns this value as the first entry of the IWORK array, and no error message related to LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

=0: Successful exit. <0: If INFO = -i, the i-th argument had an illegal value. =1: Reordering of (A, B) failed because the transformed matrix pair (A, B) would be too far from generalized Schur form; the problem is very ill-conditioned. (A, B) may have been partially reordered. If requested, 0 is returned in DIF(*), PL and PR.

Related Information

For this routine in other precisions, please see *ctgsen*, *stgsen* and *ztgsen*. It also exists with a native C interface as *LAPACKE_dtgsen*.

4.14.31 dtgsna

dtgsna estimates reciprocal condition numbers for specified eigenvalues and/or eigenvectors of a matrix pair (A, B) in generalized real Schur canonical form (or of any matrix pair ($Q*A*Z^T$, $Q*B*Z^T$) with orthogonal matrices Q and Z, where Z^T denotes the transpose of Z.

(A, B) must be in generalized real Schur form (as returned by DGGES), i.e. A is block upper triangular with 1-by-1 and 2-by-2 diagonal blocks. B is upper triangular.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtgsna(JOB, HOWMNY, SELECT, N, A, LDA, B, LDB, VL, LDVL, VR, LDVR,
                  S, DIF, MM, M, WORK, LWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dtgsna_(const char *job, const char *howmny, const armpl_int_t *select,
             const armpl_int_t *n, const double *a, const armpl_int_t *lda,
             const double *b, const armpl_int_t *ldb, const double *vl,
             const armpl_int_t *ldvl, const double *vr,
             const armpl_int_t *ldvr, double *s, double *dif,
             const armpl_int_t *mm, armpl_int_t *m, double *work,
             const armpl_int_t *lwork, armpl_int_t *iwork, armpl_int_t *info,
             ... );
```

Parameters

JOB Input parameter.

JOB is CHARACTER*1

Specifies whether condition numbers are required for eigenvalues (S) or eigenvectors (DIF): = 'E': for eigenvalues only (S); = 'V': for eigenvectors only (DIF); = 'B': for both eigenvalues and eigenvectors (S and DIF).

HOWMNY Input parameter.

HOWMNY is CHARACTER*1

= 'A': compute condition numbers for all eigenpairs; = 'S': compute condition numbers for selected eigenpairs specified by the array SELECT.

SELECT Input parameter.

SELECT is LOGICAL

SELECT is an array, dimension (N). If HOWMNY = 'S', SELECT specifies the eigenpairs for which condition numbers are required. To select condition numbers for the eigenpair corresponding to a real eigenvalue w(j), SELECT(j) must be set to .TRUE.. To select condition numbers corresponding to a complex conjugate pair of eigenvalues w(j) and w(j+1), either SELECT(j) or SELECT(j+1) or both, must be set to .TRUE.. If HOWMNY = 'A', SELECT is not referenced.

N Input parameter.

N is INTEGER

The order of the square matrix pair (A, B). N >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). The upper quasi-triangular matrix A in the pair (A, B).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). The upper triangular matrix B in the pair (A, B).

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

VL Input parameter.

VL is DOUBLE PRECISION

VL is an array, dimension (LDVL, M). If JOB = 'E' or 'B', VL must contain left eigenvectors of (A, B), corresponding to the eigenpairs specified by HOWMNY and SELECT. The eigenvectors must be stored in consecutive columns of VL, as returned by DTGEVC. If JOB = 'V', VL is not referenced.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the array VL. LDVL >= 1. If JOB = 'E' or 'B', LDVL >= N.

VR Input parameter.

VR is DOUBLE PRECISION

VR is an array, dimension (LDVR, M). If JOB = 'E' or 'B', VR must contain right eigenvectors of (A, B), corresponding to the eigenpairs specified by HOWMNY and SELECT. The eigenvectors must be stored in consecutive columns ov VR, as returned by DTGEVC. If JOB = 'V', VR is not referenced.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the array VR. LDVR >= 1. If JOB = 'E' or 'B', LDVR >= N.

S Output parameter.

S is DOUBLE PRECISION

S is an array, dimension (MM). If JOB = 'E' or 'B', the reciprocal condition numbers of the selected eigenvalues, stored in consecutive elements of the array. For a complex conjugate pair of eigenvalues two consecutive elements of S are set to the same value. Thus S(j), DIF(j), and the j-th columns of VL and VR all correspond to the same eigenpair (but not in general the j-th eigenpair, unless all eigenpairs are selected). If JOB = 'V', S is not referenced.

DIF Output parameter.

DIF is DOUBLE PRECISION

DIF is an array, dimension (MM). If JOB = 'V' or 'B', the estimated reciprocal condition numbers of the selected eigenvectors, stored in consecutive elements of the array. For a complex eigenvector two consecutive elements of DIF are set to the same value. If the eigenvalues cannot be reordered to compute DIF(j), DIF(j) is set to 0; this can only occur when the true value would be very small anyway. If JOB = 'E', DIF is not referenced.

MM Input parameter.

MM is INTEGER

The number of elements in the arrays S and DIF. MM >= M.

M Output parameter.

M is INTEGER

The number of elements of the arrays S and DIF used to store the specified condition numbers; for each selected real eigenvalue one element is used, and for each selected complex conjugate pair of eigenvalues, two elements are used. If HOWMNY = 'A', M is set to N.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N). If JOB = 'V' or 'B' LWORK >= 2*N*(N+2)+16.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (N + 6)

If JOB = 'E', IWORK is not referenced.

INFO Output parameter.

INFO is INTEGER

=0: Successful exit <0: If INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctgsna*, *stgsna* and *ztgsna*. It also exists with a native C interface as *LAPACKE_dtgsna*.

4.14.32 dtgsyl

`dtgsyl` solves the generalized Sylvester equation:

```
A * R - L * B = scale * C                (1)
D * R - L * E = scale * F
```

where R and L are unknown m-by-n matrices, (A, D), (B, E) and (C, F) are given matrix pairs of size m-by-m, n-by-n and m-by-n, respectively, with real entries. (A, D) and (B, E) must be in generalized (real) Schur canonical form, i.e. A, B are upper quasi triangular and D, E are upper triangular.

The solution (R, L) overwrites (C, F). 0 <= SCALE <= 1 is an output scaling factor chosen to avoid overflow.

In matrix notation (1) is equivalent to solve Zx = scale b, where Z is defined as

```
Z = [ kron(In, A)  -kron(B**T, Im) ]          (2)
    [ kron(In, D)  -kron(E**T, Im) ].
```

Here Ik is the identity matrix of size k and X^T is the transpose of X. kron(X, Y) is the Kronecker product between the matrices X and Y.

If TRANS = 'T', `dtgsyl` solves the transposed system Z^T *y = scale*b, which is equivalent to solve for R and L in

```
A**T * R + D**T * L = scale * C          (3)
R * B**T + L * E**T = scale * -F
```

This case (TRANS = 'T') is used to compute an one-norm-based estimate of Dif[(A,D), (B,E)], the separation between the matrix pairs (A,D) and (B,E), using DLACON.

If IJOB >= 1, `dtgsyl` computes a Frobenius norm-based estimate of Dif[(A,D),(B,E)]. That is, the reciprocal of a lower bound on the reciprocal of the smallest singular value of Z. See [1-2] for more information.

This is a level 3 BLAS algorithm.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtgsyl(TRANS, IJOB, M, N, A, LDA, B, LDB, C, LDC, D, LDD, E, LDE,
                  F, LDF, SCALE, DIF, WORK, LWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dtgsyl_(const char *trans, const armpl_int_t *ijob, const armpl_int_t *m,
             const armpl_int_t *n, const double *a, const armpl_int_t *lda,
             const double *b, const armpl_int_t *ldb, double *c,
             const armpl_int_t *ldc, const double *d, const armpl_int_t *ldd,
             const double *e, const armpl_int_t *lde, double *f,
             const armpl_int_t *ldf, double *scale, double *dif, double *work,
             const armpl_int_t *lwork, armpl_int_t *iwork, armpl_int_t *info,
             ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N', solve the generalized Sylvester equation (1). = 'T', solve the 'transposed' system (3).

IJOB Input parameter.

IJOB is INTEGER

Specifies what kind of functionality to be performed. =0: solve (1) only. =1: The functionality of 0 and 3. =2: The functionality of 0 and 4. =3: Only an estimate of Dif[(A, D), (B, E)] is computed. (look ahead strategy IJOB = 1 is used). =4: Only an estimate of Dif[(A, D), (B, E)] is computed. (DGECON on sub-systems is used). Not referenced if TRANS = 'T'.

M Input parameter.

M is INTEGER

The order of the matrices A and D, and the row dimension of the matrices C, F, R and L.

N Input parameter.

N is INTEGER

The order of the matrices B and E, and the column dimension of the matrices C, F, R and L.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, M). The upper quasi triangular matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). The upper quasi triangular matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, N). On entry, C contains the right-hand-side of the first matrix equation in (1) or (3). On exit, if IJOB = 0, 1 or 2, C has been overwritten by the solution R. If IJOB = 3 or 4 and TRANS = 'N', C holds R, the solution achieved during the computation of the Dif-estimate.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (LDD, M). The upper triangular matrix D.

LDD Input parameter.

LDD is INTEGER

The leading dimension of the array D. LDD >= max(1, M).

E Input parameter.

E is DOUBLE PRECISION

E is an array, dimension (LDE, N). The upper triangular matrix E.

LDE Input parameter.

LDE is INTEGER

The leading dimension of the array E. LDE >= max(1, N).

F Input and output parameter.

F is DOUBLE PRECISION

F is an array, dimension (LDF, N). On entry, F contains the right-hand-side of the second matrix equation in (1) or (3). On exit, if IJOB = 0, 1 or 2, F has been overwritten by the solution L. If IJOB = 3 or 4 and TRANS = 'N', F holds L, the solution achieved during the computation of the Dif-estimate.

LDF Input parameter.

LDF is INTEGER

The leading dimension of the array F. LDF >= max(1, M).

DIF Output parameter.

DIF is DOUBLE PRECISION

On exit DIF is the reciprocal of a lower bound of the reciprocal of the Dif-function, i.e. DIF is an upper bound of Dif[(A, D), (B, E)] = sigma_min(Z), where Z as in (2). IF IJOB = 0 or TRANS = 'T', DIF is not touched.

SCALE Output parameter.

SCALE is DOUBLE PRECISION

On exit SCALE is the scaling factor in (1) or (3). If 0 < SCALE < 1, C and F hold the solutions R and L, resp., to a slightly perturbed system but the input matrices A, B, D and E have not been changed. If SCALE = 0, C and F hold the solutions R and L, respectively, to the homogeneous system with C = F = 0. Normally, SCALE = 1.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK > = 1. If IJOB = 1 or 2 and TRANS = 'N', LWORK >= max(1,2*M*N).

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (M+N+6)

INFO Output parameter.

INFO is INTEGER

=0: successful exit <0: If INFO = -i, the i-th argument had an illegal value. >0: (A, D) and (B, E) have common or close eigenvalues.

Related Information

For this routine in other precisions, please see *ctgsyl*, *stgsyl* and *ztgsyl*. It also exists with a native C interface as *LAPACKE_dtgsyl*.

4.14.33 sggbak

`sggbak` forms the right or left eigenvectors of a real generalized eigenvalue problem A*x = lambda*B*x, by backward transformation on the computed eigenvectors of the balanced pair of matrices output by SGGBAL.

Syntax

Fortran specification:

```
use armpl_library

subroutine sggbak(JOB, SIDE, N, ILO, IHI, LSCALE, RSCALE, M, V, LDV, INFO)
```

C specification:

```
#include "armpl.h"

void sggbak_(const char *job, const char *side, const armpl_int_t *n,
             const armpl_int_t *ilo, const armpl_int_t *ihi,
             const float *lscale, const float *rscale, const armpl_int_t *m,
             float *v, const armpl_int_t *ldv, armpl_int_t *info, ... );
```

Parameters

JOB Input parameter.

`JOB` is `CHARACTER*1`

Specifies the type of backward transformation required: = 'N': do nothing, return immediately; = 'P': do backward transformation for permutation only; = 'S': do backward transformation for scaling only; = 'B': do backward transformations for both permutation and scaling. `JOB` must be the same as the argument `JOB` supplied to SGGBAL.

SIDE Input parameter.

`SIDE` is `CHARACTER*1`

= 'R': `V` contains right eigenvectors; = 'L': `V` contains left eigenvectors.

N Input parameter.

`N` is `INTEGER`

The number of rows of the matrix `V`. `N` >= 0.

ILO Input parameter.

`ILO` is `INTEGER`

IHI Input parameter.

`IHI` is `INTEGER`

The integers `ILO` and `IHI` determined by SGGBAL. 1 <= `ILO` <= `IHI` <= `N`, if `N` > 0; ILO=1 and IHI=0, if N=0.

LSCALE Input parameter.

LSCALE is REAL

LSCALE is an array, dimension (N). Details of the permutations and/or scaling factors applied to the left side of A and B, as returned by SGGBAL.

RSCALE Input parameter.

RSCALE is REAL

RSCALE is an array, dimension (N). Details of the permutations and/or scaling factors applied to the right side of A and B, as returned by SGGBAL.

M Input parameter.

M is INTEGER

The number of columns of the matrix V. M >= 0.

V Input and output parameter.

V is REAL

V is an array, dimension (LDV, M). On entry, the matrix of right or left eigenvectors to be transformed, as returned by STGEVC. On exit, V is overwritten by the transformed eigenvectors.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the matrix V. LDV >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cggbak*, *dggbak* and *zggbak*. It also exists with a native C interface as *LAPACKE_sggbak*.

4.14.34 sggbal

sggbal balances a pair of general real matrices (A,B). This involves, first, permuting A and B by similarity transformations to isolate eigenvalues in the first 1 to ILO$-$1 and last IHI+1 to N elements on the diagonal; and second, applying a diagonal similarity transformation to rows and columns ILO to IHI to make the rows and columns as close in norm as possible. Both steps are optional.

Balancing may reduce the 1-norm of the matrices, and improve the accuracy of the computed eigenvalues and/or eigenvectors in the generalized eigenvalue problem A*x = lambda*B*x.

Syntax

Fortran specification:

```
use armpl_library

subroutine sggbal(JOB, N, A, LDA, B, LDB, ILO, IHI, LSCALE, RSCALE, WORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void sggbal_(const char *job, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, float *b, const armpl_int_t *ldb,
             armpl_int_t *ilo, armpl_int_t *ihi, float *lscale, float *rscale,
             float *work, armpl_int_t *info, ... );
```

Parameters

JOB Input parameter.

JOB is CHARACTER*1

Specifies the operations to be performed on A and B: = 'N': none: simply set ILO = 1, IHI = N, LSCALE(I) = 1.0 and RSCALE(I) = 1.0 for i = 1,...,N. = 'P': permute only; = 'S': scale only; = 'B': both permute and scale.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the input matrix A. On exit, A is overwritten by the balanced matrix. If JOB = 'N', A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, N). On entry, the input matrix B. On exit, B is overwritten by the balanced matrix. If JOB = 'N', B is not referenced.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

ILO Output parameter.

ILO is INTEGER

IHI Output parameter.

IHI is INTEGER

ILO and IHI are set to integers such that on exit A(i,j) = 0 and B(i,j) = 0 if i > j and j = 1,...,ILO-1 or i = IHI+1,...,N. If JOB = 'N' or 'S', ILO = 1 and IHI = N.

LSCALE Output parameter.

LSCALE is REAL

LSCALE is an array, dimension (N). Details of the permutations and scaling factors applied to the left side of A and B. If P(j) is the index of the row interchanged with row j, and D(j) is the scaling factor applied to row j, then LSCALE(j) = P(j) for J = 1,...,ILO-1 = D(j) for J = ILO,...,IHI = P(j) for J = IHI+1,...,N. The order in which the interchanges are made is N to IHI+1, then 1 to ILO-1.

RSCALE Output parameter.

RSCALE is REAL

RSCALE is an array, dimension (N). Details of the permutations and scaling factors applied to the right side of A and B. If P(j) is the index of the column interchanged with column j, and D(j) is the scaling factor applied to column j, then LSCALE(j) = P(j) for J = 1,...,ILO-1 = D(j) for J = ILO,...,IHI = P(j) for J = IHI+1,...,N. The order in which the interchanges are made is N to IHI+1, then 1 to ILO-1.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (lwork). lwork must be at least max(1,6*N) when JOB = 'S' or 'B', and at least 1 when JOB = 'N' or 'P'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cggbal*, *dggbal* and *zggbal*. It also exists with a native C interface as *LAPACKE_sggbal*.

4.14.35 sgges

sgges computes for a pair of N-by-N real nonsymmetric matrices (A,B), the generalized eigenvalues, the generalized real Schur form (S,T), optionally, the left and/or right matrices of Schur vectors (VSL and VSR). This gives the generalized Schur factorization

```
(A,B) = ( (VSL)*S*(VSR)**T, (VSL)*T*(VSR)**T )
```

Optionally, it also orders the eigenvalues so that a selected cluster of eigenvalues appears in the leading diagonal blocks of the upper quasi-triangular matrix S and the upper triangular matrix T.The leading columns of VSL and VSR then form an orthonormal basis for the corresponding left and right eigenspaces (deflating subspaces).

(If only the generalized eigenvalues are needed, use the driver SGGEV instead, which is faster.)

A generalized eigenvalue for a pair of matrices (A,B) is a scalar w or a ratio alpha/beta = w, such that A - w*B is singular. It is usually represented as the pair (alpha,beta), as there is a reasonable interpretation for beta=0 or both being zero.

A pair of matrices (S,T) is in generalized real Schur form if T is upper triangular with non-negative diagonal and S is block upper triangular with 1-by-1 and 2-by-2 blocks. 1-by-1 blocks correspond to real generalized eigenvalues, while 2-by-2 blocks of S will be "standardized" by making the corresponding elements of T have the form:

```
[  a   0  ]
[  0   b  ]
```

and the pair of corresponding 2-by-2 blocks in S and T will have a complex conjugate pair of generalized eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgges(JOBVSL, JOBVSR, SORT, SELCTG, N, A, LDA, B, LDB, SDIM,
                 ALPHAR, ALPHAI, BETA, VSL, LDVSL, VSR, LDVSR, WORK, LWORK,
                 BWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgges_(const char *jobvsl, const char *jobvsr, const char *sort,
            ARMPL_SGGES_SELCTG selctg, const armpl_int_t *n, float *a,
            const armpl_int_t *lda, float *b, const armpl_int_t *ldb,
            armpl_int_t *sdim, float *alphar, float *alphai, float *beta,
            float *vsl, const armpl_int_t *ldvsl, float *vsr,
            const armpl_int_t *ldvsr, float *work, const armpl_int_t *lwork,
            armpl_int_t *bwork, armpl_int_t *info, ... );
```

Parameters

JOBVSL Input parameter.

JOBVSL is CHARACTER*1

= 'N': do not compute the left Schur vectors; = 'V': compute the left Schur vectors.

JOBVSR Input parameter.

JOBVSR is CHARACTER*1

= 'N': do not compute the right Schur vectors; = 'V': compute the right Schur vectors.

SORT Input parameter.

SORT is CHARACTER*1

Specifies whether or not to order the eigenvalues on the diagonal of the generalized Schur form. = 'N': Eigenvalues are not ordered; = 'S': Eigenvalues are ordered (see SELCTG);

SELCTG Input parameter.

SELCTG is a LOGICAL FUNCTION of three REAL arguments

SELCTG must be declared EXTERNAL in the calling subroutine. If SORT = 'N', SELCTG is not referenced. If SORT = 'S', SELCTG is used to select eigenvalues to sort to the top left of the Schur form. An eigenvalue (ALPHAR(j)+ALPHAI(j))/BETA(j) is selected if SELCTG(ALPHAR(j),ALPHAI(j),BETA(j)) is true; i.e. if either one of a complex conjugate pair of eigenvalues is selected, then both complex eigenvalues are selected.

Note that in the ill-conditioned case, a selected complex eigenvalue may no longer satisfy SELCTG(ALPHAR(j),ALPHAI(j), BETA(j)) = .TRUE. after ordering. INFO is to be set to N+2 in this case.

N Input parameter.

N is INTEGER

The order of the matrices A, B, VSL, and VSR. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the first of the pair of matrices. On exit, A has been overwritten by its generalized Schur form S.

LDA Input parameter.

LDA is INTEGER

The leading dimension of A. LDA >= max(1, N).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, N). On entry, the second of the pair of matrices. On exit, B has been overwritten by its generalized Schur form T.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B. LDB >= max(1, N).

SDIM Output parameter.

SDIM is INTEGER

If SORT = 'N', SDIM = 0. If SORT = 'S', SDIM = number of eigenvalues (after sorting) for which SELCTG is true. (Complex conjugate pairs for which SELCTG is true for either eigenvalue count as 2.)

ALPHAR Output parameter.

ALPHAR is REAL

ALPHAR is an array, dimension (N) .

ALPHAI Output parameter.

ALPHAI is REAL

ALPHAI is an array, dimension (N) .

BETA Output parameter.

BETA is REAL

BETA is an array, dimension (N). On exit, (ALPHAR(j) + ALPHAI(j)*i)/BETA(j), j=1,..., N, will be the generalized eigenvalues. ALPHAR(j) + ALPHAI(j)*i, and BETA(j),j=1,...,N are the diagonals of the complex Schur form (S,T) that would result if the 2-by-2 diagonal blocks of the real Schur form of (A, B) were further reduced to triangular form using 2-by-2 complex unitary transformations. If ALPHAI(j) is zero, then the j-th eigenvalue is real; if positive, then the j-th and (j+1)-st eigenvalues are a complex conjugate pair, with ALPHAI(j+1) negative.

Note: the quotients ALPHAR(j)/BETA(j) and ALPHAI(j)/BETA(j) may easily over- or underflow, and BETA(j) may even be zero. Thus, the user should avoid naively computing the ratio. However, ALPHAR and ALPHAI will be always less than and usually comparable with norm(A) in magnitude, and BETA always less than and usually comparable with norm(B).

VSL Output parameter.

VSL is REAL

VSL is an array, dimension (LDVSL, N). If JOBVSL = 'V', VSL will contain the left Schur vectors. Not referenced if JOBVSL = 'N'.

LDVSL Input parameter.

LDVSL is INTEGER

The leading dimension of the matrix VSL. LDVSL >=1, and if JOBVSL = 'V', LDVSL >= N.

VSR Output parameter.

VSR is REAL

VSR is an array, dimension (LDVSR, N). If JOBVSR = 'V', VSR will contain the right Schur vectors. Not referenced if JOBVSR = 'N'.

LDVSR Input parameter.

LDVSR is INTEGER

The leading dimension of the matrix VSR. LDVSR >= 1, and if JOBVSR = 'V', LDVSR >= N.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If N = 0, LWORK >= 1, else LWORK >= max(8*N,6*N+16). For good performance , LWORK must generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

BWORK Output parameter.

BWORK is LOGICAL

BWORK is an array, dimension (N). Not referenced if SORT = 'N'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. = 1,...,N: The QZ iteration failed. (A, B) are not in Schur form, but ALPHAR(j), ALPHAI(j), and BETA(j) should be correct for j=INFO+1,...,N. > N: =N+1: other than QZ iteration failed in SHGEQZ. =N+2: after reordering, roundoff changed values of some complex eigenvalues so that leading eigenvalues in the Generalized Schur form no longer satisfy SELCTG=.TRUE. This could also be caused due to scaling. =N+3: reordering failed in STGSEN.

Related Information

For this routine in other precisions, please see *cgges*, *dgges* and *zgges*. It also exists with a native C interface as *LAPACKE_sgges*.

4.14.36 sgges3

sgges3 computes for a pair of N-by-N real nonsymmetric matrices (A,B), the generalized eigenvalues, the generalized real Schur form (S,T), optionally, the left and/or right matrices of Schur vectors (VSL and VSR). This gives the generalized Schur factorization

```
(A,B) = ( (VSL)*S*(VSR)**T, (VSL)*T*(VSR)**T )
```

Optionally, it also orders the eigenvalues so that a selected cluster of eigenvalues appears in the leading diagonal blocks of the upper quasi-triangular matrix S and the upper triangular matrix T.The leading columns of VSL and VSR then form an orthonormal basis for the corresponding left and right eigenspaces (deflating subspaces).

(If only the generalized eigenvalues are needed, use the driver SGGEV instead, which is faster.)

A generalized eigenvalue for a pair of matrices (A,B) is a scalar w or a ratio alpha/beta = w, such that A - w*B is singular. It is usually represented as the pair (alpha,beta), as there is a reasonable interpretation for beta=0 or both being zero.

A pair of matrices (S,T) is in generalized real Schur form if T is upper triangular with non-negative diagonal and S is block upper triangular with 1-by-1 and 2-by-2 blocks. 1-by-1 blocks correspond to real generalized eigenvalues, while 2-by-2 blocks of S will be "standardized" by making the corresponding elements of T have the form:

```
[  a  0  ]
[  0  b  ]
```

and the pair of corresponding 2-by-2 blocks in S and T will have a complex conjugate pair of generalized eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgges3(JOBVSL, JOBVSR, SORT, SELCTG, N, A, LDA, B, LDB, SDIM,
                  ALPHAR, ALPHAI, BETA, VSL, LDVSL, VSR, LDVSR, WORK, LWORK,
                  BWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgges3_(const char *jobvsl, const char *jobvsr, const char *sort,
             ARMPL_SGGES3_SELCTG selctg, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, float *b, const armpl_int_t *ldb,
             armpl_int_t *sdim, float *alphar, float *alphai, float *beta,
             float *vsl, const armpl_int_t *ldvsl, float *vsr,
             const armpl_int_t *ldvsr, float *work, const armpl_int_t *lwork,
             armpl_int_t *bwork, armpl_int_t *info, ... );
```

Parameters

JOBVSL Input parameter.

JOBVSL is CHARACTER*1

= 'N': do not compute the left Schur vectors; = 'V': compute the left Schur vectors.

JOBVSR Input parameter.

JOBVSR is CHARACTER*1

= 'N': do not compute the right Schur vectors; = 'V': compute the right Schur vectors.

SORT Input parameter.

SORT is CHARACTER*1

Specifies whether or not to order the eigenvalues on the diagonal of the generalized Schur form. = 'N': Eigenvalues are not ordered; = 'S': Eigenvalues are ordered (see SELCTG);

SELCTG Input parameter.

SELCTG is a LOGICAL FUNCTION of three REAL arguments

SELCTG must be declared EXTERNAL in the calling subroutine. If SORT = 'N', SELCTG is not referenced. If SORT = 'S', SELCTG is used to select eigenvalues to sort to the top left of the Schur form. An eigenvalue (ALPHAR(j)+ALPHAI(j))/BETA(j) is selected if SELCTG(ALPHAR(j),ALPHAI(j),BETA(j)) is true; i.e. if either one of a complex conjugate pair of eigenvalues is selected, then both complex eigenvalues are selected.

Note that in the ill-conditioned case, a selected complex eigenvalue may no longer satisfy SELCTG(ALPHAR(j),ALPHAI(j), BETA(j)) = .TRUE. after ordering. INFO is to be set to N+2 in this case.

N Input parameter.

N is INTEGER

The order of the matrices A, B, VSL, and VSR. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the first of the pair of matrices. On exit, A has been overwritten by its generalized Schur form S.

LDA Input parameter.

LDA is INTEGER

The leading dimension of A. LDA >= max(1, N).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, N). On entry, the second of the pair of matrices. On exit, B has been overwritten by its generalized Schur form T.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B. LDB >= max(1, N).

SDIM Output parameter.

SDIM is INTEGER

If SORT = 'N', SDIM = 0. If SORT = 'S', SDIM = number of eigenvalues (after sorting) for which SELCTG is true. (Complex conjugate pairs for which SELCTG is true for either eigenvalue count as 2.)

ALPHAR Output parameter.

ALPHAR is REAL

ALPHAR is an array, dimension (N) .

ALPHAI Output parameter.

ALPHAI is REAL

ALPHAI is an array, dimension (N) .

BETA Output parameter.

BETA is REAL

BETA is an array, dimension (N). On exit, (ALPHAR(j) + ALPHAI(j)*i)/BETA(j), j=1,..., N, will be the generalized eigenvalues. ALPHAR(j) + ALPHAI(j)*i, and BETA(j),j=1,...,N are the diagonals of the complex Schur form (S,T) that would result if the 2-by-2 diagonal blocks of the real Schur form of (A, B) were further reduced to triangular form using 2-by-2 complex unitary transformations. If ALPHAI(j) is zero, then the j-th eigenvalue is real; if positive, then the j-th and (j+1)-st eigenvalues are a complex conjugate pair, with ALPHAI(j+1) negative.

Note: the quotients ALPHAR(j)/BETA(j) and ALPHAI(j)/BETA(j) may easily over- or underflow, and BETA(j) may even be zero. Thus, the user should avoid naively computing the ratio. However, ALPHAR and ALPHAI will be always less than and usually comparable with norm(A) in magnitude, and BETA always less than and usually comparable with norm(B).

VSL Output parameter.

VSL is REAL

VSL is an array, dimension (LDVSL, N). If JOBVSL = 'V', VSL will contain the left Schur vectors. Not referenced if JOBVSL = 'N'.

LDVSL Input parameter.

LDVSL is INTEGER

The leading dimension of the matrix VSL. LDVSL >=1, and if JOBVSL = 'V', LDVSL >= N.

VSR Output parameter.

VSR is REAL

VSR is an array, dimension (LDVSR, N). If JOBVSR = 'V', VSR will contain the right Schur vectors. Not referenced if JOBVSR = 'N'.

LDVSR Input parameter.

LDVSR is INTEGER

The leading dimension of the matrix VSR. LDVSR >= 1, and if JOBVSR = 'V', LDVSR >= N.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

BWORK Output parameter.

BWORK is LOGICAL

BWORK is an array, dimension (N). Not referenced if SORT = 'N'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. = 1,...,N: The QZ iteration failed. (A, B) are not in Schur form, but ALPHAR(j), ALPHAI(j), and BETA(j) should be correct for j=INFO+1,...,N. > N: =N+1: other than QZ iteration failed in SHGEQZ. =N+2: after reordering, roundoff changed values of some complex eigenvalues so that leading eigenvalues in the Generalized Schur form no longer satisfy SELCTG=.TRUE. This could also be caused due to scaling. =N+3: reordering failed in STGSEN.

Related Information

For this routine in other precisions, please see *cgges3*, *dgges3* and *zgges3*. It also exists with a native C interface as *LAPACKE_sgges3*.

4.14.37 sggesx

sggesx computes for a pair of N-by-N real nonsymmetric matrices (A,B), the generalized eigenvalues, the real Schur form (S,T), and, optionally, the left and/or right matrices of Schur vectors (VSL and VSR). This gives the generalized Schur factorization

```
(A,B) = ( (VSL) S (VSR)**T, (VSL) T (VSR)**T )
```

Optionally, it also orders the eigenvalues so that a selected cluster of eigenvalues appears in the leading diagonal blocks of the upper quasi-triangular matrix S and the upper triangular matrix T; computes a reciprocal condition number for the average of the selected eigenvalues (RCONDE); and computes a reciprocal condition number for the right and left deflating subspaces corresponding to the selected eigenvalues (RCONDV). The leading columns of VSL and VSR then form an orthonormal basis for the corresponding left and right eigenspaces (deflating subspaces).

A generalized eigenvalue for a pair of matrices (A,B) is a scalar w or a ratio alpha/beta = w, such that A - w*B is singular. It is usually represented as the pair (alpha,beta), as there is a reasonable interpretation for beta=0 or for both being zero.

A pair of matrices (S,T) is in generalized real Schur form if T is upper triangular with non-negative diagonal and S is block upper triangular with 1-by-1 and 2-by-2 blocks. 1-by-1 blocks correspond to real generalized eigenvalues, while 2-by-2 blocks of S will be "standardized" by making the corresponding elements of T have the form:

```
[  a   0  ]
[  0   b  ]
```

and the pair of corresponding 2-by-2 blocks in S and T will have a complex conjugate pair of generalized eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine sggesx(JOBVSL, JOBVSR, SORT, SELCTG, SENSE, N, A, LDA, B, LDB,
                  SDIM, ALPHAR, ALPHAI, BETA, VSL, LDVSL, VSR, LDVSR, RCONDE,
                  RCONDV, WORK, LWORK, IWORK, LIWORK, BWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sggesx_(const char *jobvsl, const char *jobvsr, const char *sort,
             ARMPL_SGGESX_SELCTG selctg, const char *sense,
             const armpl_int_t *n, float *a, const armpl_int_t *lda, float *b,
             const armpl_int_t *ldb, armpl_int_t *sdim, float *alphar,
             float *alphai, float *beta, float *vsl, const armpl_int_t *ldvsl,
             float *vsr, const armpl_int_t *ldvsr, float *rconde,
             float *rcondv, float *work, const armpl_int_t *lwork,
             armpl_int_t *iwork, const armpl_int_t *liwork,
             armpl_int_t *bwork, armpl_int_t *info, ... );
```

Parameters

JOBVSL Input parameter.

JOBVSL is CHARACTER*1

= 'N': do not compute the left Schur vectors; = 'V': compute the left Schur vectors.

JOBVSR Input parameter.

JOBVSR is CHARACTER*1

= 'N': do not compute the right Schur vectors; = 'V': compute the right Schur vectors.

SORT Input parameter.

SORT is CHARACTER*1

Specifies whether or not to order the eigenvalues on the diagonal of the generalized Schur form. = 'N': Eigenvalues are not ordered; = 'S': Eigenvalues are ordered (see SELCTG).

SELCTG Input parameter.

SELCTG is a LOGICAL FUNCTION of three REAL arguments

SELCTG must be declared EXTERNAL in the calling subroutine. If SORT = 'N', SELCTG is not referenced. If SORT = 'S', SELCTG is used to select eigenvalues to sort to the top left of the Schur form. An eigenvalue (ALPHAR(j)+ALPHAI(j))/BETA(j) is selected if SELCTG(ALPHAR(j),ALPHAI(j),BETA(j)) is true; i.e. if either one of a complex conjugate pair of eigenvalues is selected, then both complex eigenvalues are selected. Note that a selected complex eigenvalue may no longer satisfy SELCTG(ALPHAR(j),ALPHAI(j),BETA(j)) = .TRUE. after ordering, since ordering may change the value of complex eigenvalues (especially if the eigenvalue is ill-conditioned), in this case INFO is set to N+3.

SENSE Input parameter.

SENSE is CHARACTER*1

Determines which reciprocal condition numbers are computed. = 'N' : None are computed; = 'E' : Computed for average of selected eigenvalues only; = 'V' : Computed for selected deflating subspaces only; = 'B' : Computed for both. If SENSE = 'E', 'V', or 'B', SORT must equal 'S'.

N Input parameter.

N is INTEGER

The order of the matrices A, B, VSL, and VSR. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the first of the pair of matrices. On exit, A has been overwritten by its generalized Schur form S.

LDA Input parameter.

LDA is INTEGER

The leading dimension of A. LDA >= max(1, N).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, N). On entry, the second of the pair of matrices. On exit, B has been overwritten by its generalized Schur form T.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B. LDB >= max(1, N).

SDIM Output parameter.

SDIM is INTEGER

If SORT = 'N', SDIM = 0. If SORT = 'S', SDIM = number of eigenvalues (after sorting) for which SELCTG is true. (Complex conjugate pairs for which SELCTG is true for either eigenvalue count as 2.)

ALPHAR Output parameter.

ALPHAR is REAL

ALPHAR is an array, dimension (N) .

ALPHAI Output parameter.

ALPHAI is REAL

ALPHAI is an array, dimension (N) .

BETA Output parameter.

BETA is REAL

BETA is an array, dimension (N). On exit, (ALPHAR(j) + ALPHAI(j)*i)/BETA(j), j=1,..., N, will be the generalized eigenvalues. ALPHAR(j) + ALPHAI(j)*i and BETA(j),j=1,...,N are the diagonals of the complex Schur form (S,T) that would result if the 2-by-2 diagonal blocks of the real Schur form of (A, B) were further reduced to triangular form using 2-by-2 complex unitary transformations. If ALPHAI(j) is zero, then the j-th eigenvalue is real; if positive, then the j-th and (j+1)-st eigenvalues are a complex conjugate pair, with ALPHAI(j+1) negative.

Note: the quotients ALPHAR(j)/BETA(j) and ALPHAI(j)/BETA(j) may easily over- or underflow, and BETA(j) may even be zero. Thus, the user should avoid naively computing the ratio. However, ALPHAR and ALPHAI will be always less than and usually comparable with norm(A) in magnitude, and BETA always less than and usually comparable with norm(B).

VSL Output parameter.

VSL is REAL

VSL is an array, dimension (LDVSL, N). If JOBVSL = 'V', VSL will contain the left Schur vectors. Not referenced if JOBVSL = 'N'.

LDVSL Input parameter.

LDVSL is INTEGER

The leading dimension of the matrix VSL. LDVSL >=1, and if JOBVSL = 'V', LDVSL >= N.

VSR Output parameter.

VSR is REAL

VSR is an array, dimension (LDVSR, N). If JOBVSR = 'V', VSR will contain the right Schur vectors. Not referenced if JOBVSR = 'N'.

LDVSR Input parameter.

LDVSR is INTEGER

The leading dimension of the matrix VSR. LDVSR >= 1, and if JOBVSR = 'V', LDVSR >= N.

RCONDE Output parameter.

RCONDE is REAL

RCONDE is an array, dimension (2). If SENSE = 'E' or 'B', RCONDE(1) and RCONDE(2) contain the reciprocal condition numbers for the average of the selected eigenvalues. Not referenced if SENSE = 'N' or 'V'.

RCONDV Output parameter.

RCONDV is REAL

RCONDV is an array, dimension (2). If SENSE = 'V' or 'B', RCONDV(1) and RCONDV(2) contain the reciprocal condition numbers for the selected deflating subspaces. Not referenced if SENSE = 'N' or 'E'.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If N = 0, LWORK >= 1, else if SENSE = 'E', 'V', or 'B', LWORK >= max(8*N, 6*N+16, 2*SDIM*(N-SDIM)), else LWORK >= max(8*N, 6*N+16). Note that 2*SDIM*(N-SDIM) <= N*N/2. Note also that an error is only returned if LWORK < max(8*N, 6*N+16), but if SENSE = 'E' or 'V' or 'B' this may not be large enough.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the bound on the optimal size of the WORK array and the minimum size of the IWORK array, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the minimum LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. If SENSE = 'N' or N = 0, LIWORK >= 1, otherwise LIWORK >= N+6.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the bound on the optimal size of the WORK array and the minimum size of the IWORK array, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

BWORK Output parameter.

BWORK is LOGICAL

BWORK is an array, dimension (N). Not referenced if SORT = 'N'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. = 1,...,N: The QZ iteration failed. (A, B) are not in Schur form, but ALPHAR(j), ALPHAI(j), and BETA(j) should be correct for j=INFO+1,...,N. > N: =N+1: other than QZ iteration failed in SHGEQZ =N+2: after reordering, roundoff changed values of some complex eigenvalues so that leading eigenvalues in the Generalized Schur form no longer satisfy SELCTG=.TRUE. This could also be caused due to scaling. =N+3: reordering failed in STGSEN.

Related Information

For this routine in other precisions, please see *cggesx*, *dggesx* and *zggesx*. It also exists with a native C interface as *LAPACKE_sggesx*.

4.14.38 sggev

sggev computes for a pair of N-by-N real nonsymmetric matrices (A,B) the generalized eigenvalues, and optionally, the left and/or right generalized eigenvectors.

A generalized eigenvalue for a pair of matrices (A,B) is a scalar lambda or a ratio alpha/beta = lambda, such that A - lambda*B is singular. It is usually represented as the pair (alpha,beta), as there is a reasonable interpretation for beta=0, and even for both being zero.

The right eigenvector v(j) corresponding to the eigenvalue lambda(j) of (A,B) satisfies

```
A * v(j) = lambda(j) * B * v(j).
```

The left eigenvector u(j) corresponding to the eigenvalue lambda(j) of (A,B) satisfies

```
u(j)**H * A  = lambda(j) * u(j)**H * B .
```

where $u(j)^H$ is the conjugate-transpose of u(j).

Syntax

Fortran specification:

```
use armpl_library

subroutine sggev(JOBVL, JOBVR, N, A, LDA, B, LDB, ALPHAR, ALPHAI, BETA, VL,
                 LDVL, VR, LDVR, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sggev_(const char *jobvl, const char *jobvr, const armpl_int_t *n,
            float *a, const armpl_int_t *lda, float *b,
            const armpl_int_t *ldb, float *alphar, float *alphai, float *beta,
            float *vl, const armpl_int_t *ldvl, float *vr,
            const armpl_int_t *ldvr, float *work, const armpl_int_t *lwork,
            armpl_int_t *info, ... );
```

Parameters

JOBVL Input parameter.

JOBVL is CHARACTER*1

= 'N': do not compute the left generalized eigenvectors; = 'V': compute the left generalized eigenvectors.

JOBVR Input parameter.

JOBVR is CHARACTER*1

= 'N': do not compute the right generalized eigenvectors; = 'V': compute the right generalized eigenvectors.

N Input parameter.

N is INTEGER

The order of the matrices A, B, VL, and VR. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the matrix A in the pair (A, B). On exit, A has been overwritten.

LDA Input parameter.

LDA is INTEGER

The leading dimension of A. LDA >= max(1, N).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, N). On entry, the matrix B in the pair (A, B). On exit, B has been overwritten.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B. LDB >= max(1, N).

ALPHAR Output parameter.

ALPHAR is REAL

ALPHAR is an array, dimension (N) .

ALPHAI Output parameter.

ALPHAI is REAL

ALPHAI is an array, dimension (N) .

BETA Output parameter.

BETA is REAL

BETA is an array, dimension (N). On exit, (ALPHAR(j) + ALPHAI(j)*i)/BETA(j), j=1,..., N, will be the generalized eigenvalues. If ALPHAI(j) is zero, then the j-th eigenvalue is real; if positive, then the j-th and (j+1)-st eigenvalues are a complex conjugate pair, with ALPHAI(j+1) negative.

Note: the quotients ALPHAR(j)/BETA(j) and ALPHAI(j)/BETA(j) may easily over- or underflow, and BETA(j) may even be zero. Thus, the user should avoid naively computing the ratio alpha/beta. However, ALPHAR and ALPHAI will be always less than and usually comparable with norm(A) in magnitude, and BETA always less than and usually comparable with norm(B).

VL Output parameter.

VL is REAL

VL is an array, dimension (LDVL, N). If JOBVL = 'V', the left eigenvectors u(j) are stored one after another in the columns of VL, in the same order as their eigenvalues. If the j-th eigenvalue is real, then u(j) = VL(:,j), the j-th column of VL. If the j-th and (j+1)-th eigenvalues form a complex conjugate pair, then u(j) = VL(:,j)+i*VL(:,j+1) and u(j+1) = VL(:,j)-i*VL(:,j+1). Each eigenvector is scaled so the largest component has abs(real part)+abs(imag. part)=1. Not referenced if JOBVL = 'N'.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the matrix VL. LDVL >= 1, and if JOBVL = 'V', LDVL >= N.

VR Output parameter.

VR is REAL

VR is an array, dimension (LDVR, N). If JOBVR = 'V', the right eigenvectors v(j) are stored one after another in the columns of VR, in the same order as their eigenvalues. If the j-th eigenvalue is real, then v(j) = VR(:,j), the j-th column of VR. If the j-th and (j+1)-th eigenvalues form a complex conjugate pair, then v(j) = VR(:,j)+i*VR(:,j+1) and v(j+1) = VR(:,j)-i*VR(:,j+1). Each eigenvector is scaled so the largest component has abs(real part)+abs(imag. part)=1. Not referenced if JOBVR = 'N'.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the matrix VR. LDVR >= 1, and if JOBVR = 'V', LDVR >= N.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,8*N). For good performance, LWORK must generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. = 1,...,N: The QZ iteration failed. No eigenvectors have been calculated, but ALPHAR(j), ALPHAI(j), and BETA(j) should be correct for j=INFO+1,...,N. > N: =N+1: other than QZ iteration failed in SHGEQZ. =N+2: error return from STGEVC.

Related Information

For this routine in other precisions, please see *cggev*, *dggev* and *zggev*. It also exists with a native C interface as *LAPACKE_sggev*.

4.14.39 sggev3

sggev3 computes for a pair of N-by-N real nonsymmetric matrices (A,B) the generalized eigenvalues, and optionally, the left and/or right generalized eigenvectors.

A generalized eigenvalue for a pair of matrices (A,B) is a scalar lambda or a ratio alpha/beta = lambda, such that A - lambda*B is singular. It is usually represented as the pair (alpha,beta), as there is a reasonable interpretation for beta=0, and even for both being zero.

The right eigenvector v(j) corresponding to the eigenvalue lambda(j) of (A,B) satisfies

```
A * v(j) = lambda(j) * B * v(j).
```

The left eigenvector u(j) corresponding to the eigenvalue lambda(j) of (A,B) satisfies

```
u(j)**H * A  = lambda(j) * u(j)**H * B .
```

where $u(j)^H$ is the conjugate-transpose of u(j).

Syntax

Fortran specification:

```
use armpl_library

subroutine sggev3(JOBVL, JOBVR, N, A, LDA, B, LDB, ALPHAR, ALPHAI, BETA, VL,
                  LDVL, VR, LDVR, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sggev3_(const char *jobvl, const char *jobvr, const armpl_int_t *n,
             float *a, const armpl_int_t *lda, float *b,
             const armpl_int_t *ldb, float *alphar, float *alphai,
             float *beta, float *vl, const armpl_int_t *ldvl, float *vr,
             const armpl_int_t *ldvr, float *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

JOBVL Input parameter.

JOBVL is CHARACTER*1

= 'N': do not compute the left generalized eigenvectors; = 'V': compute the left generalized eigenvectors.

JOBVR Input parameter.

JOBVR is CHARACTER*1

= 'N': do not compute the right generalized eigenvectors; = 'V': compute the right generalized eigenvectors.

N Input parameter.

N is INTEGER

The order of the matrices A, B, VL, and VR. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the matrix A in the pair (A, B). On exit, A has been overwritten.

LDA Input parameter.

LDA is INTEGER

The leading dimension of A. LDA >= max(1, N).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, N). On entry, the matrix B in the pair (A, B). On exit, B has been overwritten.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B. LDB >= max(1, N).

ALPHAR Output parameter.

ALPHAR is REAL

ALPHAR is an array, dimension (N) .

ALPHAI Output parameter.

ALPHAI is REAL

ALPHAI is an array, dimension (N) .

BETA Output parameter.

BETA is REAL

BETA is an array, dimension (N). On exit, (ALPHAR(j) + ALPHAI(j)*i)/BETA(j), j=1,..., N, will be the generalized eigenvalues. If ALPHAI(j) is zero, then the j-th eigenvalue is real; if positive, then the j-th and (j+1)-st eigenvalues are a complex conjugate pair, with ALPHAI(j+1) negative.

Note: the quotients ALPHAR(j)/BETA(j) and ALPHAI(j)/BETA(j) may easily over- or underflow, and BETA(j) may even be zero. Thus, the user should avoid naively computing the ratio alpha/beta. However, ALPHAR and ALPHAI will be always less than and usually comparable with norm(A) in magnitude, and BETA always less than and usually comparable with norm(B).

VL Output parameter.

VL is REAL

VL is an array, dimension (LDVL, N). If JOBVL = 'V', the left eigenvectors u(j) are stored one after another in the columns of VL, in the same order as their eigenvalues. If the j-th eigenvalue is real, then u(j) = VL(:,j), the j-th column of VL. If the j-th and (j+1)-th eigenvalues form a complex conjugate pair, then u(j) = VL(:,j)+i*VL(:,j+1) and u(j+1) = VL(:,j)-i*VL(:,j+1). Each eigenvector is scaled so the largest component has abs(real part)+abs(imag. part)=1. Not referenced if JOBVL = 'N'.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the matrix VL. LDVL >= 1, and if JOBVL = 'V', LDVL >= N.

VR Output parameter.

VR is REAL

VR is an array, dimension (LDVR, N). If JOBVR = 'V', the right eigenvectors v(j) are stored one after another in the columns of VR, in the same order as their eigenvalues. If the j-th eigenvalue is real, then v(j) = VR(:,j), the j-th column of VR. If the j-th and (j+1)-th eigenvalues form a complex conjugate pair, then v(j) = VR(:,j)+i*VR(:,j+1) and v(j+1) = VR(:,j)-i*VR(:,j+1). Each eigenvector is scaled so the largest component has abs(real part)+abs(imag. part)=1. Not referenced if JOBVR = 'N'.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the matrix VR. LDVR >= 1, and if JOBVR = 'V', LDVR >= N.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. = 1,...,N: The QZ iteration failed. No eigenvectors have been calculated, but ALPHAR(j), ALPHAI(j), and BETA(j) should be correct for j=INFO+1,...,N. > N: =N+1: other than QZ iteration failed in SHGEQZ. =N+2: error return from STGEVC.

Related Information

For this routine in other precisions, please see *cggev3*, *dggev3* and *zggev3*. It also exists with a native C interface as *LAPACKE_sggev3*.

4.14.40 sggevx

sggevx computes for a pair of N-by-N real nonsymmetric matrices (A,B) the generalized eigenvalues, and optionally, the left and/or right generalized eigenvectors.

Optionally also, it computes a balancing transformation to improve the conditioning of the eigenvalues and eigenvectors (ILO, IHI, LSCALE, RSCALE, ABNRM, and BBNRM), reciprocal condition numbers for the eigenvalues (RCONDE), and reciprocal condition numbers for the right eigenvectors (RCONDV).

A generalized eigenvalue for a pair of matrices (A,B) is a scalar lambda or a ratio alpha/beta = lambda, such that A - lambda*B is singular. It is usually represented as the pair (alpha,beta), as there is a reasonable interpretation for beta=0, and even for both being zero.

The right eigenvector v(j) corresponding to the eigenvalue lambda(j) of (A,B) satisfies

```
A * v(j) = lambda(j) * B * v(j) .
```

The left eigenvector u(j) corresponding to the eigenvalue lambda(j) of (A,B) satisfies

```
u(j)**H * A  = lambda(j) * u(j)**H * B.
```

where $u(j)^H$ is the conjugate-transpose of u(j).

Syntax

Fortran specification:

```
use armpl_library

subroutine sggevx(BALANC, JOBVL, JOBVR, SENSE, N, A, LDA, B, LDB, ALPHAR,
                  ALPHAI, BETA, VL, LDVL, VR, LDVR, ILO, IHI, LSCALE, RSCALE,
                  ABNRM, BBNRM, RCONDE, RCONDV, WORK, LWORK, IWORK, BWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void sggevx_(const char *balanc, const char *jobvl, const char *jobvr,
             const char *sense, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, float *b, const armpl_int_t *ldb,
             float *alphar, float *alphai, float *beta, float *vl,
             const armpl_int_t *ldvl, float *vr, const armpl_int_t *ldvr,
             armpl_int_t *ilo, armpl_int_t *ihi, float *lscale, float *rscale,
             float *abnrm, float *bbnrm, float *rconde, float *rcondv,
             float *work, const armpl_int_t *lwork, armpl_int_t *iwork,
             armpl_int_t *bwork, armpl_int_t *info, ... );
```

Parameters

BALANC Input parameter.

BALANC is CHARACTER*1

Specifies the balance option to be performed. = 'N': do not diagonally scale or permute; = 'P': permute only; = 'S': scale only; = 'B': both permute and scale. Computed reciprocal condition numbers will be for the matrices after permuting and/or balancing. Permuting does not change condition numbers (in exact arithmetic), but balancing does.

JOBVL Input parameter.

JOBVL is CHARACTER*1

= 'N': do not compute the left generalized eigenvectors; = 'V': compute the left generalized eigenvectors.

JOBVR Input parameter.

JOBVR is CHARACTER*1

= 'N': do not compute the right generalized eigenvectors; = 'V': compute the right generalized eigenvectors.

SENSE Input parameter.

SENSE is CHARACTER*1

Determines which reciprocal condition numbers are computed. = 'N': none are computed; = 'E': computed for eigenvalues only; = 'V': computed for eigenvectors only; = 'B': computed for eigenvalues and eigenvectors.

N Input parameter.

N is INTEGER

The order of the matrices A, B, VL, and VR. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the matrix A in the pair (A, B). On exit, A has been overwritten. If JOBVL='V' or JOBVR='V' or both, then A contains the first part of the real Schur form of the "balanced" versions of the input A and B.

LDA Input parameter.

LDA is INTEGER

The leading dimension of A. LDA >= max(1, N).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, N). On entry, the matrix B in the pair (A, B). On exit, B has been overwritten. If JOBVL='V' or JOBVR='V' or both, then B contains the second part of the real Schur form of the "balanced" versions of the input A and B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B. LDB >= max(1, N).

ALPHAR Output parameter.

ALPHAR is REAL

ALPHAR is an array, dimension (N) .

ALPHAI Output parameter.

ALPHAI is REAL

ALPHAI is an array, dimension (N) .

BETA Output parameter.

BETA is REAL

BETA is an array, dimension (N). On exit, (ALPHAR(j) + ALPHAI(j)*i)/BETA(j), j=1,..., N, will be the generalized eigenvalues. If ALPHAI(j) is zero, then the j-th eigenvalue is real; if positive, then the j-th and (j+1)-st eigenvalues are a complex conjugate pair, with ALPHAI(j+1) negative.

Note: the quotients ALPHAR(j)/BETA(j) and ALPHAI(j)/BETA(j) may easily over- or underflow, and BETA(j) may even be zero. Thus, the user should avoid naively computing the ratio ALPHA/BETA. However, ALPHAR and ALPHAI will be always less than and usually comparable with norm(A) in magnitude, and BETA always less than and usually comparable with norm(B).

VL Output parameter.

VL is REAL

VL is an array, dimension (LDVL, N). If JOBVL = 'V', the left eigenvectors u(j) are stored one after another in the columns of VL, in the same order as their eigenvalues. If the j-th eigenvalue is real, then u(j) = VL(:,j), the j-th column of VL. If the j-th and (j+1)-th eigenvalues form a complex conjugate pair, then u(j) = VL(:,j)+i*VL(:,j+1) and u(j+1) = VL(:,j)-i*VL(:,j+1). Each eigenvector will be scaled so the largest component have abs(real part) + abs(imag. part) = 1. Not referenced if JOBVL = 'N'.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the matrix VL. LDVL >= 1, and if JOBVL = 'V', LDVL >= N.

VR Output parameter.

VR is REAL

VR is an array, dimension (LDVR, N). If JOBVR = 'V', the right eigenvectors v(j) are stored one after another in the columns of VR, in the same order as their eigenvalues. If the j-th eigenvalue is real, then v(j) = VR(:,j), the j-th column of VR. If the j-th and (j+1)-th eigenvalues form a complex conjugate pair, then v(j) = VR(:,j)+i*VR(:,j+1) and v(j+1) = VR(:,j)-i*VR(:,j+1). Each eigenvector will be scaled so the largest component have abs(real part) + abs(imag. part) = 1. Not referenced if JOBVR = 'N'.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the matrix VR. LDVR >= 1, and if JOBVR = 'V', LDVR >= N.

ILO Output parameter.

ILO is INTEGER

IHI Output parameter.

IHI is INTEGER

ILO and IHI are integer values such that on exit A(i,j) = 0 and B(i,j) = 0 if i > j and j = 1,...,ILO-1 or i = IHI+1,...,N. If BALANC = 'N' or 'S', ILO = 1 and IHI = N.

LSCALE Output parameter.

LSCALE is REAL

LSCALE is an array, dimension (N). Details of the permutations and scaling factors applied to the left side of A and B. If PL(j) is the index of the row interchanged with row j, and DL(j) is the scaling factor applied to row j, then LSCALE(j) = PL(j) for j = 1,...,ILO-1 = DL(j) for j = ILO,...,IHI = PL(j) for j = IHI+1,...,N. The order in which the interchanges are made is N to IHI+1, then 1 to ILO-1.

RSCALE Output parameter.

RSCALE is REAL

RSCALE is an array, dimension (N). Details of the permutations and scaling factors applied to the right side of A and B. If PR(j) is the index of the column interchanged with column j, and DR(j) is the scaling factor applied to column j, then RSCALE(j) = PR(j) for j = 1,...,ILO-1 = DR(j) for j = ILO,...,IHI = PR(j) for j = IHI+1,...,N The order in which the interchanges are made is N to IHI+1, then 1 to ILO-1.

ABNRM Output parameter.

ABNRM is REAL

The one-norm of the balanced matrix A.

BBNRM Output parameter.

BBNRM is REAL

The one-norm of the balanced matrix B.

RCONDE Output parameter.

RCONDE is REAL

RCONDE is an array, dimension (N). If SENSE = 'E' or 'B', the reciprocal condition numbers of the eigenvalues, stored in consecutive elements of the array. For a complex conjugate pair of eigenvalues two consecutive elements of RCONDE are set to the same value. Thus RCONDE(j), RCONDV(j), and the j-th columns of VL and VR all correspond to the j-th eigenpair. If SENSE = 'N' or 'V', RCONDE is not referenced.

RCONDV Output parameter.

RCONDV is REAL

RCONDV is an array, dimension (N). If SENSE = 'V' or 'B', the estimated reciprocal condition numbers of the eigenvectors, stored in consecutive elements of the array. For a complex eigenvector two consecutive elements of RCONDV are set to the same value. If the eigenvalues cannot be reordered to compute RCONDV(j), RCONDV(j) is set to 0; this can only occur when the true value would be very small anyway. If SENSE = 'N' or 'E', RCONDV is not referenced.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,2*N). If BALANC = 'S' or 'B', or JOBVL = 'V', or JOBVR = 'V', LWORK >= max(1,6*N). If SENSE = 'E', LWORK >= max(1,10*N). If SENSE = 'V' or 'B', LWORK >= 2*N*N+8*N+16.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (N+6)

If SENSE = 'E', IWORK is not referenced.

BWORK Output parameter.

BWORK is LOGICAL

BWORK is an array, dimension (N). If SENSE = 'N', BWORK is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. = 1,...,N: The QZ iteration failed. No eigenvectors have been calculated, but ALPHAR(j), ALPHAI(j), and BETA(j) should be correct for j=INFO+1,...,N. > N: =N+1: other than QZ iteration failed in SHGEQZ. =N+2: error return from STGEVC.

Related Information

For this routine in other precisions, please see *cggevx*, *dggevx* and *zggevx*. It also exists with a native C interface as *LAPACKE_sggevx*.

4.14.41 sgghd3

sgghd3 reduces a pair of real matrices (A,B) to generalized upper Hessenberg form using orthogonal transformations, where A is a general matrix and B is upper triangular. The form of the generalized eigenvalue problem is

```
A*x = lambda*B*x,
```

and B is typically made upper triangular by computing its QR factorization and moving the orthogonal matrix Q to the left side of the equation.

This subroutine simultaneously reduces A to a Hessenberg matrix H:

```
Q**T*A*Z = H
```

and transforms B to another upper triangular matrix T:

```
Q**T*B*Z = T
```

in order to reduce the problem to its standard form

```
H*y = lambda*T*y
```

where y = Z^T *x.

The orthogonal matrices Q and Z are determined as products of Givens rotations. They may either be formed explicitly, or they may be postmultiplied into input matrices Q1 and Z1, so that

```
Q1 * A * Z1**T = (Q1*Q) * H * (Z1*Z)**T

Q1 * B * Z1**T = (Q1*Q) * T * (Z1*Z)**T
```

If Q1 is the orthogonal matrix from the QR factorization of B in the original equation A*x = lambda*B*x, then `sgghd3` reduces the original problem to generalized Hessenberg form.

This is a blocked variant of SGGHRD, using matrix-matrix multiplications for parts of the computation to enhance performance.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgghd3(COMPQ, COMPZ, N, ILO, IHI, A, LDA, B, LDB, Q, LDQ, Z, LDZ,
                  WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgghd3_(const char *compq, const char *compz, const armpl_int_t *n,
             const armpl_int_t *ilo, const armpl_int_t *ihi, float *a,
             const armpl_int_t *lda, float *b, const armpl_int_t *ldb,
             float *q, const armpl_int_t *ldq, float *z,
             const armpl_int_t *ldz, float *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

COMPQ Input parameter.

`COMPQ` is `CHARACTER*1`

= 'N': do not compute Q; = 'I': `Q` is initialized to the unit matrix, and the orthogonal matrix `Q` is returned; = 'V': `Q` must contain an orthogonal matrix Q1 on entry, and the product Q1* `Q` is returned.

COMPZ Input parameter.

`COMPZ` is `CHARACTER*1`

= 'N': do not compute Z; = 'I': `Z` is initialized to the unit matrix, and the orthogonal matrix `Z` is returned; = 'V': `Z` must contain an orthogonal matrix Z1 on entry, and the product Z1* `Z` is returned.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

ILO and IHI mark the rows and columns of A which are to be reduced. It is assumed that A is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N. ILO and IHI are normally set by a previous call to SGGBAL; otherwise they should be set to 1 and N respectively. 1 <= ILO <= IHI <= N, if N > 0; ILO=1 and IHI=0, if N=0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the N-by-N general matrix to be reduced. On exit, the upper triangle and the first subdiagonal of A are overwritten with the upper Hessenberg matrix H, and the rest is set to zero.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, N). On entry, the N-by-N upper triangular matrix B. On exit, the upper triangular matrix T = Q^T B Z. The elements below the diagonal are set to zero.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Q Input and output parameter.

Q is REAL

Q is an array, dimension (LDQ, N). On entry, if COMPQ = 'V', the orthogonal matrix Q1, typically from the QR factorization of B. On exit, if COMPQ='I', the orthogonal matrix Q, and if COMPQ = 'V', the product Q1*Q. Not referenced if COMPQ='N'.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= N if COMPQ='V' or 'I'; LDQ >= 1 otherwise.

Z Input and output parameter.

Z is REAL

Z is an array, dimension (LDZ, N). On entry, if COMPZ = 'V', the orthogonal matrix Z1. On exit, if COMPZ='I', the orthogonal matrix Z, and if COMPZ = 'V', the product Z1*Z. Not referenced if COMPZ='N'.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= N if COMPZ='V' or 'I'; LDZ >= 1 otherwise.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (LWORK). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= 1. For optimum performance LWORK >= 6*N*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cgghd3*, *dgghd3* and *zgghd3*. It also exists with a native C interface as *LAPACKE_sgghd3*.

4.14.42 sgghrd

sgghrd reduces a pair of real matrices (A,B) to generalized upper Hessenberg form using orthogonal transformations, where A is a general matrix and B is upper triangular. The form of the generalized eigenvalue problem is

```
A*x = lambda*B*x,
```

and B is typically made upper triangular by computing its QR factorization and moving the orthogonal matrix Q to the left side of the equation.

This subroutine simultaneously reduces A to a Hessenberg matrix H:

```
Q**T*A*Z = H
```

and transforms B to another upper triangular matrix T:

```
Q**T*B*Z = T
```

in order to reduce the problem to its standard form

```
H*y = lambda*T*y
```

where y = Z^T *x.

The orthogonal matrices Q and Z are determined as products of Givens rotations. They may either be formed explicitly, or they may be postmultiplied into input matrices Q1 and Z1, so that

```
Q1 * A * Z1**T = (Q1*Q) * H * (Z1*Z)**T

Q1 * B * Z1**T = (Q1*Q) * T * (Z1*Z)**T
```

If Q1 is the orthogonal matrix from the QR factorization of B in the original equation A*x = lambda*B*x, then sgghrd reduces the original problem to generalized Hessenberg form.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgghrd(COMPQ, COMPZ, N, ILO, IHI, A, LDA, B, LDB, Q, LDQ, Z, LDZ,
                  INFO)
```

C specification:

```
#include "armpl.h"

void sgghrd_(const char *compq, const char *compz, const armpl_int_t *n,
             const armpl_int_t *ilo, const armpl_int_t *ihi, float *a,
             const armpl_int_t *lda, float *b, const armpl_int_t *ldb,
             float *q, const armpl_int_t *ldq, float *z,
             const armpl_int_t *ldz, armpl_int_t *info, ... );
```

Parameters

COMPQ Input parameter.

COMPQ is CHARACTER*1

= 'N': do not compute Q; = 'I': Q is initialized to the unit matrix, and the orthogonal matrix Q is returned; = 'V': Q must contain an orthogonal matrix Q1 on entry, and the product Q1* Q is returned.

COMPZ Input parameter.

COMPZ is CHARACTER*1

= 'N': do not compute Z; = 'I': Z is initialized to the unit matrix, and the orthogonal matrix Z is returned; = 'V': Z must contain an orthogonal matrix Z1 on entry, and the product Z1* Z is returned.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

ILO and IHI mark the rows and columns of A which are to be reduced. It is assumed that A is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N. ILO and IHI are normally set by a previous call to SGGBAL; otherwise they should be set to 1 and N respectively. 1 <= ILO <= IHI <= N, if N > 0; ILO=1 and IHI=0, if N=0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the N-by-N general matrix to be reduced. On exit, the upper triangle and the first subdiagonal of A are overwritten with the upper Hessenberg matrix H, and the rest is set to zero.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, N). On entry, the N-by-N upper triangular matrix B. On exit, the upper triangular matrix T = Q^T B Z. The elements below the diagonal are set to zero.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Q Input and output parameter.

Q is REAL

Q is an array, dimension (LDQ, N). On entry, if COMPQ = 'V', the orthogonal matrix Q1, typically from the QR factorization of B. On exit, if COMPQ='I', the orthogonal matrix Q, and if COMPQ = 'V', the product Q1*Q. Not referenced if COMPQ='N'.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= N if COMPQ='V' or 'I'; LDQ >= 1 otherwise.

Z Input and output parameter.

Z is REAL

Z is an array, dimension (LDZ, N). On entry, if COMPZ = 'V', the orthogonal matrix Z1. On exit, if COMPZ='I', the orthogonal matrix Z, and if COMPZ = 'V', the product Z1*Z. Not referenced if COMPZ='N'.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= N if COMPZ='V' or 'I'; LDZ >= 1 otherwise.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cgghrd*, *dgghrd* and *zgghrd*. It also exists with a native C interface as *LAPACKE_sgghrd*.

4.14.43 shgeqz

shgeqz computes the eigenvalues of a real matrix pair (H,T), where H is an upper Hessenberg matrix and T is upper triangular, using the double-shift QZ method. Matrix pairs of this type are produced by the reduction to generalized upper Hessenberg form of a real matrix pair (A,B):

```
A = Q1*H*Z1**T,  B = Q1*T*Z1**T,
```

as computed by SGGHRD.

If JOB='S', then the Hessenberg-triangular pair (H,T) is also reduced to generalized Schur form,

```
H = Q*S*Z**T,  T = Q*P*Z**T,
```

where Q and Z are orthogonal matrices, P is an upper triangular matrix, and S is a quasi-triangular matrix with 1-by-1 and 2-by-2 diagonal blocks.

The 1-by-1 blocks correspond to real eigenvalues of the matrix pair (H,T) and the 2-by-2 blocks correspond to complex conjugate pairs of eigenvalues.

Additionally, the 2-by-2 upper triangular diagonal blocks of P corresponding to 2-by-2 blocks of S are reduced to positive diagonal form, i.e., if S(j+1,j) is non-zero, then P(j+1,j) = P(j,j+1) = 0, P(j,j) > 0, and P(j+1,j+1) > 0.

Optionally, the orthogonal matrix Q from the generalized Schur factorization may be postmultiplied into an input matrix Q1, and the orthogonal matrix Z may be postmultiplied into an input matrix Z1. If Q1 and Z1 are the orthogonal matrices from SGGHRD that reduced the matrix pair (A,B) to generalized upper Hessenberg form, then the output matrices Q1*Q and Z1*Z are the orthogonal factors from the generalized Schur factorization of (A,B):

```
A = (Q1*Q)*S*(Z1*Z)**T,  B = (Q1*Q)*P*(Z1*Z)**T.
```

To avoid overflow, eigenvalues of the matrix pair (H,T) (equivalently, of (A,B)) are computed as a pair of values (alpha,beta), where alpha is complex and beta real. If beta is nonzero, lambda = alpha / beta is an eigenvalue of the generalized nonsymmetric eigenvalue problem (GNEP)

```
A*x = lambda*B*x
```

and if alpha is nonzero, mu = beta / alpha is an eigenvalue of the alternate form of the GNEP

```
mu*A*y = B*y.
```

Real eigenvalues can be read directly from the generalized Schur form:

```
alpha = S(i,i), beta = P(i,i).
```

Ref: C.B. Moler & G.W. Stewart, “An Algorithm for Generalized Matrix Eigenvalue Problems”, SIAM J. Numer. Anal., 10(1973), pp. 241–256.

Syntax

Fortran specification:

```
use armpl_library

subroutine shgeqz(JOB, COMPQ, COMPZ, N, ILO, IHI, H, LDH, T, LDT, ALPHAR,
                  ALPHAI, BETA, Q, LDQ, Z, LDZ, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void shgeqz_(const char *job, const char *compq, const char *compz,
             const armpl_int_t *n, const armpl_int_t *ilo,
             const armpl_int_t *ihi, float *h, const armpl_int_t *ldh,
             float *t, const armpl_int_t *ldt, float *alphar, float *alphai,
             float *beta, float *q, const armpl_int_t *ldq, float *z,
             const armpl_int_t *ldz, float *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

JOB Input parameter.

JOB is CHARACTER*1

= ‘E’: Compute eigenvalues only; = ‘S’: Compute eigenvalues and the Schur form.

COMPQ Input parameter.

COMPQ is CHARACTER*1

= 'N': Left Schur vectors (Q) are not computed; = 'I': Q is initialized to the unit matrix and the matrix Q of left Schur vectors of (H, T) is returned; = 'V': Q must contain an orthogonal matrix Q1 on entry and the product Q1* Q is returned.

COMPZ Input parameter.

COMPZ is CHARACTER*1

= 'N': Right Schur vectors (Z) are not computed; = 'I': Z is initialized to the unit matrix and the matrix Z of right Schur vectors of (H, T) is returned; = 'V': Z must contain an orthogonal matrix Z1 on entry and the product Z1* Z is returned.

N Input parameter.

N is INTEGER

The order of the matrices H, T, Q, and Z. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

ILO and IHI mark the rows and columns of H which are in Hessenberg form. It is assumed that A is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N. If N > 0, 1 <= ILO <= IHI <= N; if N = 0, ILO=1 and IHI=0.

H Input and output parameter.

H is REAL

H is an array, dimension (LDH, N). On entry, the N-by-N upper Hessenberg matrix H. On exit, if JOB = 'S', H contains the upper quasi-triangular matrix S from the generalized Schur factorization. If JOB = 'E', the diagonal blocks of H match those of S, but the rest of H is unspecified.

LDH Input parameter.

LDH is INTEGER

The leading dimension of the array H. LDH >= max(1, N).

T Input and output parameter.

T is REAL

T is an array, dimension (LDT, N). On entry, the N-by-N upper triangular matrix T. On exit, if JOB = 'S', T contains the upper triangular matrix P from the generalized Schur factorization; 2-by-2 diagonal blocks of P corresponding to 2-by-2 blocks of S are reduced to positive diagonal form, i.e., if H(j+1,j) is non-zero, then T(j+1,j) = T(j,j+1) = 0, T(j,j) > 0, and T(j+1,j+1) > 0. If JOB = 'E', the diagonal blocks of T match those of P, but the rest of T is unspecified.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

ALPHAR Output parameter.

ALPHAR is REAL

ALPHAR is an array, dimension (N). The real parts of each scalar alpha defining an eigenvalue of GNEP.

ALPHAI Output parameter.

ALPHAI is REAL

ALPHAI is an array, dimension (N). The imaginary parts of each scalar alpha defining an eigenvalue of GNEP. If ALPHAI(j) is zero, then the j-th eigenvalue is real; if positive, then the j-th and (j+1)-st eigenvalues are a complex conjugate pair, with ALPHAI(j+1) = -ALPHAI(j).

BETA Output parameter.

BETA is REAL

BETA is an array, dimension (N). The scalars beta that define the eigenvalues of GNEP. Together, the quantities alpha = (ALPHAR(j),ALPHAI(j)) and beta = BETA(j) represent the j-th eigenvalue of the matrix pair (A,B), in one of the forms lambda = alpha/beta or mu = beta/alpha. Since either lambda or mu may overflow, they should not, in general, be computed.

Q Input and output parameter.

Q is REAL

Q is an array, dimension (LDQ, N). On entry, if COMPQ = 'V', the orthogonal matrix Q1 used in the reduction of (A,B) to generalized Hessenberg form. On exit, if COMPQ = 'I', the orthogonal matrix of left Schur vectors of (H, T), and if COMPQ = 'V', the orthogonal matrix of left Schur vectors of (A,B). Not referenced if COMPQ = 'N'.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= 1. If COMPQ='V' or 'I', then LDQ >= N.

Z Input and output parameter.

Z is REAL

Z is an array, dimension (LDZ, N). On entry, if COMPZ = 'V', the orthogonal matrix Z1 used in the reduction of (A,B) to generalized Hessenberg form. On exit, if COMPZ = 'I', the orthogonal matrix of right Schur vectors of (H, T), and if COMPZ = 'V', the orthogonal matrix of right Schur vectors of (A,B). Not referenced if COMPZ = 'N'.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1. If COMPZ='V' or 'I', then LDZ >= N.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO >= 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N).

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value = 1,...,N: the QZ iteration did not converge. (H, T) is not in Schur form, but ALPHAR(i), ALPHAI(i), and BETA(i), i=INFO+1,...,N should be correct. = N+1,...,2*N: the shift calculation failed. (H, T) is not in Schur form, but ALPHAR(i), ALPHAI(i), and BETA(i), i=INFO-N+1,...,N should be correct.

Related Information

For this routine in other precisions, please see *chgeqz*, *dhgeqz* and *zhgeqz*. It also exists with a native C interface as *LAPACKE_shgeqz*.

4.14.44 stgevc

`stgevc` computes some or all of the right and/or left eigenvectors of a pair of real matrices (S,P), where S is a quasi-triangular matrix and P is upper triangular. Matrix pairs of this type are produced by the generalized Schur factorization of a matrix pair (A,B):

```
A = Q*S*Z**T,  B = Q*P*Z**T
```

as computed by SGGHRD + SHGEQZ.

The right eigenvector x and the left eigenvector y of (S,P) corresponding to an eigenvalue w are defined by:

```
S*x = w*P*x,  (y**H)*S = w*(y**H)*P,
```

where y^H denotes the conjugate tranpose of y. The eigenvalues are not input to this routine, but are computed directly from the diagonal blocks of S and P.

This routine returns the matrices X and/or Y of right and left eigenvectors of (S,P), or the products Z*X and/or Q*Y, where Z and Q are input matrices. If Q and Z are the orthogonal factors from the generalized Schur factorization of a matrix pair (A,B), then Z*X and Q*Y are the matrices of right and left eigenvectors of (A,B).

Syntax

Fortran specification:

```
use armpl_library

subroutine stgevc(SIDE, HOWMNY, SELECT, N, S, LDS, P, LDP, VL, LDVL, VR, LDVR,
                  MM, M, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void stgevc_(const char *side, const char *howmny, const armpl_int_t *select,
             const armpl_int_t *n, const float *s, const armpl_int_t *lds,
             const float *p, const armpl_int_t *ldp, float *vl,
             const armpl_int_t *ldvl, float *vr, const armpl_int_t *ldvr,
             const armpl_int_t *mm, armpl_int_t *m, float *work,
             armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

`SIDE` is `CHARACTER*1`

= 'R': compute right eigenvectors only; = 'L': compute left eigenvectors only; = 'B': compute both right and left eigenvectors.

HOWMNY Input parameter.

`HOWMNY` is `CHARACTER*1`

= ‘A’: compute all right and/or left eigenvectors; = ‘B’: compute all right and/or left eigenvectors, backtransformed by the matrices in VR and/or VL; = ‘S’: compute selected right and/or left eigenvectors, specified by the logical array SELECT.

SELECT Input parameter.

SELECT is LOGICAL

SELECT is an array, dimension (N). If HOWMNY=’S’, SELECT specifies the eigenvectors to be computed. If w(j) is a real eigenvalue, the corresponding real eigenvector is computed if SELECT(j) is .TRUE.. If w(j) and w(j+1) are the real and imaginary parts of a complex eigenvalue, the corresponding complex eigenvector is computed if either SELECT(j) or SELECT(j+1) is .TRUE., and on exit SELECT(j) is set to .TRUE. and SELECT(j+1) is set to .FALSE.. Not referenced if HOWMNY = ‘A’ or ‘B’.

N Input parameter.

N is INTEGER

The order of the matrices S and P. N >= 0.

S Input parameter.

S is REAL

S is an array, dimension (LDS, N). The upper quasi-triangular matrix S from a generalized Schur factorization, as computed by SHGEQZ.

LDS Input parameter.

LDS is INTEGER

The leading dimension of array S. LDS >= max(1, N).

P Input parameter.

P is REAL

P is an array, dimension (LDP, N). The upper triangular matrix P from a generalized Schur factorization, as computed by SHGEQZ. 2-by-2 diagonal blocks of P corresponding to 2-by-2 blocks of S must be in positive diagonal form.

LDP Input parameter.

LDP is INTEGER

The leading dimension of array P. LDP >= max(1, N).

VL Input and output parameter.

VL is REAL

VL is an array, dimension (LDVL, MM). On entry, if SIDE = ‘L’ or ‘B’ and HOWMNY = ‘B’, VL must contain an N-by-N matrix Q (usually the orthogonal matrix Q of left Schur vectors returned by SHGEQZ). On exit, if SIDE = ‘L’ or ‘B’, VL contains: if HOWMNY = ‘A’, the matrix Y of left eigenvectors of (S, P); if HOWMNY = ‘B’, the matrix Q*Y; if HOWMNY = ‘S’, the left eigenvectors of (S, P) specified by SELECT, stored consecutively in the columns of VL, in the same order as their eigenvalues.

A complex eigenvector corresponding to a complex eigenvalue is stored in two consecutive columns, the first holding the real part, and the second the imaginary part.

Not referenced if SIDE = ‘R’.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of array VL. LDVL >= 1, and if SIDE = ‘L’ or ‘B’, LDVL >= N.

VR Input and output parameter.

VR is REAL

VR is an array, dimension (LDVR, MM). On entry, if SIDE = 'R' or 'B' and HOWMNY = 'B', VR must contain an N-by-N matrix Z (usually the orthogonal matrix Z of right Schur vectors returned by SHGEQZ).

On exit, if SIDE = 'R' or 'B', VR contains: if HOWMNY = 'A', the matrix X of right eigenvectors of (S, P); if HOWMNY = 'B' or 'b', the matrix Z*X; if HOWMNY = 'S' or 's', the right eigenvectors of (S, P) specified by SELECT, stored consecutively in the columns of VR, in the same order as their eigenvalues.

A complex eigenvector corresponding to a complex eigenvalue is stored in two consecutive columns, the first holding the real part and the second the imaginary part.

Not referenced if SIDE = 'L'.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the array VR. LDVR >= 1, and if SIDE = 'R' or 'B', LDVR >= N.

MM Input parameter.

MM is INTEGER

The number of columns in the arrays VL and/or VR. MM >= M.

M Output parameter.

M is INTEGER

The number of columns in the arrays VL and/or VR actually used to store the eigenvectors. If HOWMNY = 'A' or 'B', M is set to N. Each selected real eigenvector occupies one column and each selected complex eigenvector occupies two columns.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (6*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: the 2-by-2 block (INFO:INFO+1) does not have a complex eigenvalue.

Related Information

For this routine in other precisions, please see *ctgevc*, *dtgevc* and *ztgevc*. It also exists with a native C interface as *LAPACKE_stgevc*.

4.14.45 stgexc

stgexc reorders the generalized real Schur decomposition of a real matrix pair (A,B) using an orthogonal equivalence transformation

```
(A, B) = Q * (A, B) * Z**T,
```

so that the diagonal block of (A, B) with row index IFST is moved to row ILST.

(A, B) must be in generalized real Schur canonical form (as returned by SGGES), i.e. A is block upper triangular with 1-by-1 and 2-by-2 diagonal blocks. B is upper triangular.

Optionally, the matrices Q and Z of generalized Schur vectors are updated.

```
Q(in) * A(in) * Z(in)**T = Q(out) * A(out) * Z(out)**T
Q(in) * B(in) * Z(in)**T = Q(out) * B(out) * Z(out)**T
```

Syntax

Fortran specification:

```
use armpl_library

subroutine stgexc(WANTQ, WANTZ, N, A, LDA, B, LDB, Q, LDQ, Z, LDZ, IFST, ILST,
                  WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void stgexc_(const armpl_int_t *wantq, const armpl_int_t *wantz,
             const armpl_int_t *n, float *a, const armpl_int_t *lda, float *b,
             const armpl_int_t *ldb, float *q, const armpl_int_t *ldq,
             float *z, const armpl_int_t *ldz, armpl_int_t *ifst,
             armpl_int_t *ilst, float *work, const armpl_int_t *lwork,
             armpl_int_t *info);
```

Parameters

WANTQ Input parameter.

WANTQ is LOGICAL

.TRUE. : update the left transformation matrix Q; .FALSE.: do not update Q.

WANTZ Input parameter.

WANTZ is LOGICAL

.TRUE. : update the right transformation matrix Z; .FALSE.: do not update Z.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the matrix A in generalized real Schur canonical form. On exit, the updated matrix A, again in generalized real Schur canonical form.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, N). On entry, the matrix B in generalized real Schur canonical form (A, B). On exit, the updated matrix B, again in generalized real Schur canonical form (A, B).

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Q Input and output parameter.

Q is REAL

Q is an array, dimension (LDQ, N). On entry, if WANTQ = .TRUE., the orthogonal matrix Q. On exit, the updated matrix Q. If WANTQ = .FALSE., Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= 1. If WANTQ = .TRUE., LDQ >= N.

Z Input and output parameter.

Z is REAL

Z is an array, dimension (LDZ, N). On entry, if WANTZ = .TRUE., the orthogonal matrix Z. On exit, the updated matrix Z. If WANTZ = .FALSE., Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1. If WANTZ = .TRUE., LDZ >= N.

IFST Input and output parameter.

IFST is INTEGER

ILST Input and output parameter.

ILST is INTEGER

Specify the reordering of the diagonal blocks of (A, B). The block with row index IFST is moved to row ILST, by a sequence of swapping between adjacent blocks. On exit, if IFST pointed on entry to the second row of a 2-by-2 block, it is changed to point to the first row; ILST always points to the first row of the block in its final position (which may differ from its input value by +1 or -1). 1 <= IFST, ILST <= N.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= 1 when N <= 1, otherwise LWORK >= 4* N + 16.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

=0: successful exit. <0: if INFO = -i, the i-th argument had an illegal value. =1: The transformed matrix pair (A, B) would be too far from generalized Schur form; the problem is ill- conditioned. (A, B) may have been partially reordered, and ILST points to the first row of the current position of the block being moved.

Related Information

For this routine in other precisions, please see *ctgexc*, *dtgexc* and *ztgexc*. It also exists with a native C interface as *LAPACKE_stgexc*.

4.14.46 stgsen

`stgsen` reorders the generalized real Schur decomposition of a real matrix pair (A, B) (in terms of an orthonormal equivalence trans- formation Q^T * (A, B) * Z), so that a selected cluster of eigenvalues appears in the leading diagonal blocks of the upper quasi-triangular matrix A and the upper triangular B. The leading columns of Q and Z form orthonormal bases of the corresponding left and right eigen- spaces (deflating subspaces). (A, B) must be in generalized real Schur canonical form (as returned by SGGES), i.e. A is block upper triangular with 1-by-1 and 2-by-2 diagonal blocks. B is upper triangular.

`stgsen` also computes the generalized eigenvalues

```
w(j) = (ALPHAR(j) + i*ALPHAI(j))/BETA(j)
```

of the reordered matrix pair (A, B).

Optionally, `stgsen` computes the estimates of reciprocal condition numbers for eigenvalues and eigenspaces. These are Difu[(A11,B11), (A22,B22)] and Difl[(A11,B11), (A22,B22)], i.e. the separation(s) between the matrix pairs (A11, B11) and (A22,B22) that correspond to the selected cluster and the eigenvalues outside the cluster, resp., and norms of "projections" onto left and right eigenspaces w.r.t. the selected cluster in the (1,1)-block.

Syntax

Fortran specification:

```
use armpl_library

subroutine stgsen(IJOB, WANTQ, WANTZ, SELECT, N, A, LDA, B, LDB, ALPHAR,
                  ALPHAI, BETA, Q, LDQ, Z, LDZ, M, PL, PR, DIF, WORK, LWORK,
                  IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void stgsen_(const armpl_int_t *ijob, const armpl_int_t *wantq,
             const armpl_int_t *wantz, const armpl_int_t *select,
             const armpl_int_t *n, float *a, const armpl_int_t *lda, float *b,
             const armpl_int_t *ldb, float *alphar, float *alphai,
             float *beta, float *q, const armpl_int_t *ldq, float *z,
             const armpl_int_t *ldz, armpl_int_t *m, float *pl, float *pr,
             float *dif, float *work, const armpl_int_t *lwork,
             armpl_int_t *iwork, const armpl_int_t *liwork,
             armpl_int_t *info);
```

Parameters

IJOB Input parameter.

`IJOB` is `INTEGER`

Specifies whether condition numbers are required for the cluster of eigenvalues (PL and PR) or the deflating subspaces (Difu and Difl): =0: Only reorder w.r.t. `SELECT`. No extras. =1: Reciprocal of norms of "projections" onto left and right eigenspaces w.r.t. the selected cluster (PL and PR). =2: Upper bounds on Difu and Difl. F-norm-based estimate (DIF(1:2)). =3: Estimate of Difu and Difl. 1-norm-based estimate (DIF(1:2)). About 5 times as expensive as `IJOB` = 2. =4: Compute `PL`, `PR` and `DIF` (i.e. 0, 1 and 2 above): Economic version to get it all. =5: Compute `PL`, `PR` and `DIF` (i.e. 0, 1 and 3 above)

WANTQ Input parameter.

`WANTQ` is `LOGICAL`

.TRUE. : update the left transformation matrix Q; .FALSE.: do not update `Q`.

WANTZ Input parameter.

WANTZ is LOGICAL

.TRUE. : update the right transformation matrix Z; .FALSE.: do not update Z.

SELECT Input parameter.

SELECT is LOGICAL

SELECT is an array, dimension (N). SELECT specifies the eigenvalues in the selected cluster. To select a real eigenvalue w(j), SELECT(j) must be set to .TRUE.. To select a complex conjugate pair of eigenvalues w(j) and w(j+1), corresponding to a 2-by-2 diagonal block, either SELECT(j) or SELECT(j+1) or both must be set to .TRUE.; a complex conjugate pair of eigenvalues must be either both included in the cluster or both excluded.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension(LDA, N). On entry, the upper quasi-triangular matrix A, with (A, B) in generalized real Schur canonical form. On exit, A is overwritten by the reordered matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is REAL

B is an array, dimension(LDB, N). On entry, the upper triangular matrix B, with (A, B) in generalized real Schur canonical form. On exit, B is overwritten by the reordered matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

ALPHAR Output parameter.

ALPHAR is REAL

ALPHAR is an array, dimension (N) .

ALPHAI Output parameter.

ALPHAI is REAL

ALPHAI is an array, dimension (N) .

BETA Output parameter.

BETA is REAL

BETA is an array, dimension (N). On exit, (ALPHAR(j) + ALPHAI(j)*i)/BETA(j), j=1,..., N, will be the generalized eigenvalues. ALPHAR(j) + ALPHAI(j)*i and BETA(j),j=1,...,N are the diagonals of the complex Schur form (S,T) that would result if the 2-by-2 diagonal blocks of the real generalized Schur form of (A, B) were further reduced to triangular form using complex unitary transformations. If ALPHAI(j) is zero, then the j-th eigenvalue is real; if positive, then the j-th and (j+1)-st eigenvalues are a complex conjugate pair, with ALPHAI(j+1) negative.

Q Input and output parameter.

Q is REAL

Q is an array, dimension (LDQ, N). On entry, if WANTQ = .TRUE., Q is an N-by-N matrix. On exit, Q has been postmultiplied by the left orthogonal transformation matrix which reorder (A, B); The leading M columns of Q form orthonormal bases for the specified pair of left eigenspaces (deflating subspaces). If WANTQ = .FALSE., Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= 1; and if WANTQ = .TRUE., LDQ >= N.

Z Input and output parameter.

Z is REAL

Z is an array, dimension (LDZ, N). On entry, if WANTZ = .TRUE., Z is an N-by-N matrix. On exit, Z has been postmultiplied by the left orthogonal transformation matrix which reorder (A, B); The leading M columns of Z form orthonormal bases for the specified pair of left eigenspaces (deflating subspaces). If WANTZ = .FALSE., Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1; If WANTZ = .TRUE., LDZ >= N.

M Output parameter.

M is INTEGER

The dimension of the specified pair of left and right eigen- spaces (deflating subspaces). 0 <= M <= N.

PL Output parameter.

PL is REAL

PR Output parameter.

PR is REAL

If IJOB = 1, 4 or 5, PL, PR are lower bounds on the reciprocal of the norm of "projections" onto left and right eigenspaces with respect to the selected cluster. 0 < PL, PR <= 1. If M = 0 or M = N, PL = PR = 1. If IJOB = 0, 2 or 3, PL and PR are not referenced.

DIF Output parameter.

DIF is REAL

DIF is an array, dimension (2).. If IJOB >= 2, DIF(1:2) store the estimates of Difu and Difl. If IJOB = 2 or 4, DIF(1:2) are F-norm-based upper bounds on Difu and Difl. If IJOB = 3 or 5, DIF(1:2) are 1-norm-based estimates of Difu and Difl. If M = 0 or N, DIF(1:2) = F-norm([A, B]). If IJOB = 0 or 1, DIF is not referenced.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= 4*N+16. If IJOB = 1, 2 or 4, LWORK >= MAX(4*N+16, 2*M*(N-M)). If IJOB = 3 or 5, LWORK >= MAX(4*N+16, 4*M*(N-M)).

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. LIWORK >= 1. If IJOB = 1, 2 or 4, LIWORK >= N+6. If IJOB = 3 or 5, LIWORK >= MAX(2*M*(N-M), N+6).

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the IWORK array, returns this value as the first entry of the IWORK array, and no error message related to LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

=0: Successful exit. <0: If INFO = -i, the i-th argument had an illegal value. =1: Reordering of (A, B) failed because the transformed matrix pair (A, B) would be too far from generalized Schur form; the problem is very ill-conditioned. (A, B) may have been partially reordered. If requested, 0 is returned in DIF(*), PL and PR.

Related Information

For this routine in other precisions, please see *ctgsen*, *dtgsen* and *ztgsen*. It also exists with a native C interface as *LAPACKE_stgsen*.

4.14.47 stgsna

stgsna estimates reciprocal condition numbers for specified eigenvalues and/or eigenvectors of a matrix pair (A, B) in generalized real Schur canonical form (or of any matrix pair ($Q*A*Z^T$, $Q*B*Z^T$) with orthogonal matrices Q and Z, where Z^T denotes the transpose of Z.

(A, B) must be in generalized real Schur form (as returned by SGGES), i.e. A is block upper triangular with 1-by-1 and 2-by-2 diagonal blocks. B is upper triangular.

Syntax

Fortran specification:

```
use armpl_library

subroutine stgsna(JOB, HOWMNY, SELECT, N, A, LDA, B, LDB, VL, LDVL, VR, LDVR,
                  S, DIF, MM, M, WORK, LWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void stgsna_(const char *job, const char *howmny, const armpl_int_t *select,
             const armpl_int_t *n, const float *a, const armpl_int_t *lda,
             const float *b, const armpl_int_t *ldb, const float *vl,
             const armpl_int_t *ldvl, const float *vr,
             const armpl_int_t *ldvr, float *s, float *dif,
             const armpl_int_t *mm, armpl_int_t *m, float *work,
             const armpl_int_t *lwork, armpl_int_t *iwork, armpl_int_t *info,
             ... );
```

Parameters

JOB Input parameter.

JOB is CHARACTER*1

Specifies whether condition numbers are required for eigenvalues (S) or eigenvectors (DIF): = 'E': for eigenvalues only (S); = 'V': for eigenvectors only (DIF); = 'B': for both eigenvalues and eigenvectors (S and DIF).

HOWMNY Input parameter.

HOWMNY is CHARACTER*1

= 'A': compute condition numbers for all eigenpairs; = 'S': compute condition numbers for selected eigenpairs specified by the array SELECT.

SELECT Input parameter.

SELECT is LOGICAL

SELECT is an array, dimension (N). If HOWMNY = 'S', SELECT specifies the eigenpairs for which condition numbers are required. To select condition numbers for the eigenpair corresponding to a real eigenvalue w(j), SELECT(j) must be set to .TRUE.. To select condition numbers corresponding to a complex conjugate pair of eigenvalues w(j) and w(j+1), either SELECT(j) or SELECT(j+1) or both, must be set to .TRUE.. If HOWMNY = 'A', SELECT is not referenced.

N Input parameter.

N is INTEGER

The order of the square matrix pair (A, B). N >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). The upper quasi-triangular matrix A in the pair (A, B).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input parameter.

B is REAL

B is an array, dimension (LDB, N). The upper triangular matrix B in the pair (A, B).

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

VL Input parameter.

VL is REAL

VL is an array, dimension (LDVL, M). If JOB = 'E' or 'B', VL must contain left eigenvectors of (A, B), corresponding to the eigenpairs specified by HOWMNY and SELECT. The eigenvectors must be stored in consecutive columns of VL, as returned by STGEVC. If JOB = 'V', VL is not referenced.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the array VL. LDVL >= 1. If JOB = 'E' or 'B', LDVL >= N.

VR Input parameter.

VR is REAL

VR is an array, dimension (LDVR, M). If JOB = 'E' or 'B', VR must contain right eigenvectors of (A, B), corresponding to the eigenpairs specified by HOWMNY and SELECT. The eigenvectors must be stored in consecutive columns ov VR, as returned by STGEVC. If JOB = 'V', VR is not referenced.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the array VR. LDVR >= 1. If JOB = 'E' or 'B', LDVR >= N.

S Output parameter.

S is REAL

S is an array, dimension (MM). If JOB = 'E' or 'B', the reciprocal condition numbers of the selected eigenvalues, stored in consecutive elements of the array. For a complex conjugate pair of eigenvalues two consecutive elements of S are set to the same value. Thus S(j), DIF(j), and the j-th columns of VL and VR all correspond to the same eigenpair (but not in general the j-th eigenpair, unless all eigenpairs are selected). If JOB = 'V', S is not referenced.

DIF Output parameter.

DIF is REAL

DIF is an array, dimension (MM). If JOB = 'V' or 'B', the estimated reciprocal condition numbers of the selected eigenvectors, stored in consecutive elements of the array. For a complex eigenvector two consecutive elements of DIF are set to the same value. If the eigenvalues cannot be reordered to compute DIF(j), DIF(j) is set to 0; this can only occur when the true value would be very small anyway. If JOB = 'E', DIF is not referenced.

MM Input parameter.

MM is INTEGER

The number of elements in the arrays S and DIF. MM >= M.

M Output parameter.

M is INTEGER

The number of elements of the arrays S and DIF used to store the specified condition numbers; for each selected real eigenvalue one element is used, and for each selected complex conjugate pair of eigenvalues, two elements are used. If HOWMNY = 'A', M is set to N.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N). If JOB = 'V' or 'B' LWORK >= 2*N*(N+2)+16.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (N + 6)

If JOB = 'E', IWORK is not referenced.

INFO Output parameter.

INFO is INTEGER

=0: Successful exit <0: If INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctgsna*, *dtgsna* and *ztgsna*. It also exists with a native C interface as *LAPACKE_stgsna*.

4.14.48 stgsyl

stgsyl solves the generalized Sylvester equation:

```
A * R - L * B = scale * C                (1)
D * R - L * E = scale * F
```

where R and L are unknown m-by-n matrices, (A, D), (B, E) and (C, F) are given matrix pairs of size m-by-m, n-by-n and m-by-n, respectively, with real entries. (A, D) and (B, E) must be in generalized (real) Schur canonical form, i.e. A, B are upper quasi triangular and D, E are upper triangular.

The solution (R, L) overwrites (C, F). 0 <= SCALE <= 1 is an output scaling factor chosen to avoid overflow.

In matrix notation (1) is equivalent to solve Zx = scale b, where Z is defined as

```
Z = [ kron(In, A)  -kron(B**T, Im) ]          (2)
    [ kron(In, D)  -kron(E**T, Im) ].
```

Here Ik is the identity matrix of size k and X^T is the transpose of X. kron(X, Y) is the Kronecker product between the matrices X and Y.

If TRANS = 'T', stgsyl solves the transposed system Z^T *y = scale*b, which is equivalent to solve for R and L in

```
A**T * R + D**T * L = scale * C          (3)
R * B**T + L * E**T = scale * -F
```

This case (TRANS = 'T') is used to compute an one-norm-based estimate of Dif[(A,D), (B,E)], the separation between the matrix pairs (A,D) and (B,E), using SLACON.

If IJOB >= 1, stgsyl computes a Frobenius norm-based estimate of Dif[(A,D),(B,E)]. That is, the reciprocal of a lower bound on the reciprocal of the smallest singular value of Z. See [1-2] for more information.

This is a level 3 BLAS algorithm.

Syntax

Fortran specification:

```
use armpl_library

subroutine stgsyl(TRANS, IJOB, M, N, A, LDA, B, LDB, C, LDC, D, LDD, E, LDE,
                  F, LDF, SCALE, DIF, WORK, LWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void stgsyl_(const char *trans, const armpl_int_t *ijob, const armpl_int_t *m,
             const armpl_int_t *n, const float *a, const armpl_int_t *lda,
```

(continues on next page)

(continued from previous page)

```
                const float *b, const armpl_int_t *ldb, float *c,
                const armpl_int_t *ldc, const float *d, const armpl_int_t *ldd,
                const float *e, const armpl_int_t *lde, float *f,
                const armpl_int_t *ldf, float *scale, float *dif, float *work,
                const armpl_int_t *lwork, armpl_int_t *iwork, armpl_int_t *info,
                ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N', solve the generalized Sylvester equation (1). = 'T', solve the 'transposed' system (3).

IJOB Input parameter.

IJOB is INTEGER

Specifies what kind of functionality to be performed. =0: solve (1) only. =1: The functionality of 0 and 3. =2: The functionality of 0 and 4. =3: Only an estimate of Dif[(A, D), (B, E)] is computed. (look ahead strategy IJOB = 1 is used). =4: Only an estimate of Dif[(A, D), (B, E)] is computed. (SGECON on sub-systems is used). Not referenced if TRANS = 'T'.

M Input parameter.

M is INTEGER

The order of the matrices A and D, and the row dimension of the matrices C, F, R and L.

N Input parameter.

N is INTEGER

The order of the matrices B and E, and the column dimension of the matrices C, F, R and L.

A Input parameter.

A is REAL

A is an array, dimension (LDA, M). The upper quasi triangular matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input parameter.

B is REAL

B is an array, dimension (LDB, N). The upper quasi triangular matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, N). On entry, C contains the right-hand-side of the first matrix equation in (1) or (3). On exit, if IJOB = 0, 1 or 2, C has been overwritten by the solution R. If IJOB = 3 or 4 and TRANS = 'N', C holds R, the solution achieved during the computation of the Dif-estimate.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

D Input parameter.

D is REAL

D is an array, dimension (LDD, M). The upper triangular matrix D.

LDD Input parameter.

LDD is INTEGER

The leading dimension of the array D. LDD >= max(1, M).

E Input parameter.

E is REAL

E is an array, dimension (LDE, N). The upper triangular matrix E.

LDE Input parameter.

LDE is INTEGER

The leading dimension of the array E. LDE >= max(1, N).

F Input and output parameter.

F is REAL

F is an array, dimension (LDF, N). On entry, F contains the right-hand-side of the second matrix equation in (1) or (3). On exit, if IJOB = 0, 1 or 2, F has been overwritten by the solution L. If IJOB = 3 or 4 and TRANS = 'N', F holds L, the solution achieved during the computation of the Dif-estimate.

LDF Input parameter.

LDF is INTEGER

The leading dimension of the array F. LDF >= max(1, M).

DIF Output parameter.

DIF is REAL

On exit DIF is the reciprocal of a lower bound of the reciprocal of the Dif-function, i.e. DIF is an upper bound of Dif[(A, D), (B, E)] = sigma_min(Z), where Z as in (2). IF IJOB = 0 or TRANS = 'T', DIF is not touched.

SCALE Output parameter.

SCALE is REAL

On exit SCALE is the scaling factor in (1) or (3). If 0 < SCALE < 1, C and F hold the solutions R and L, resp., to a slightly perturbed system but the input matrices A, B, D and E have not been changed. If SCALE = 0, C and F hold the solutions R and L, respectively, to the homogeneous system with C = F = 0. Normally, SCALE = 1.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK > = 1. If IJOB = 1 or 2 and TRANS = 'N', LWORK >= max(1,2*M*N).

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (M+N+6)

INFO Output parameter.

INFO is INTEGER

=0: successful exit <0: If INFO = -i, the i-th argument had an illegal value. >0: (A, D) and (B, E) have common or close eigenvalues.

Related Information

For this routine in other precisions, please see *ctgsyl*, *dtgsyl* and *ztgsyl*. It also exists with a native C interface as *LAPACKE_stgsyl*.

4.14.49 zggbak

zggbak forms the right or left eigenvectors of a complex generalized eigenvalue problem A*x = lambda*B*x, by backward transformation on the computed eigenvectors of the balanced pair of matrices output by ZGGBAL.

Syntax

Fortran specification:

```
use armpl_library

subroutine zggbak(JOB, SIDE, N, ILO, IHI, LSCALE, RSCALE, M, V, LDV, INFO)
```

C specification:

```
#include "armpl.h"

void zggbak_(const char *job, const char *side, const armpl_int_t *n,
             const armpl_int_t *ilo, const armpl_int_t *ihi,
             const double *lscale, const double *rscale, const armpl_int_t *m,
             armpl_doublecomplex_t *v, const armpl_int_t *ldv,
             armpl_int_t *info, ... );
```

Parameters

JOB Input parameter.

JOB is CHARACTER*1

Specifies the type of backward transformation required: = 'N': do nothing, return immediately; = 'P': do backward transformation for permutation only; = 'S': do backward transformation for scaling only; = 'B': do backward transformations for both permutation and scaling. JOB must be the same as the argument JOB supplied to ZGGBAL.

SIDE Input parameter.

SIDE is CHARACTER*1

= 'R': V contains right eigenvectors; = 'L': V contains left eigenvectors.

N Input parameter.

N is INTEGER

The number of rows of the matrix V. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

The integers ILO and IHI determined by ZGGBAL. 1 <= ILO <= IHI <= N, if N > 0; ILO=1 and IHI=0, if N=0.

LSCALE Input parameter.

LSCALE is DOUBLE PRECISION

LSCALE is an array, dimension (N). Details of the permutations and/or scaling factors applied to the left side of A and B, as returned by ZGGBAL.

RSCALE Input parameter.

RSCALE is DOUBLE PRECISION

RSCALE is an array, dimension (N). Details of the permutations and/or scaling factors applied to the right side of A and B, as returned by ZGGBAL.

M Input parameter.

M is INTEGER

The number of columns of the matrix V. M >= 0.

V Input and output parameter.

V is COMPLEX*16

V is an array, dimension (LDV, M). On entry, the matrix of right or left eigenvectors to be transformed, as returned by ZTGEVC. On exit, V is overwritten by the transformed eigenvectors.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the matrix V. LDV >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cggbak*, *dggbak* and *sggbak*. It also exists with a native C interface as *LAPACKE_zggbak*.

4.14.50 zggbal

zggbal balances a pair of general complex matrices (A,B). This involves, first, permuting A and B by similarity transformations to isolate eigenvalues in the first 1 to ILO$-$1 and last IHI+1 to N elements on the diagonal; and second, applying a diagonal similarity transformation to rows and columns ILO to IHI to make the rows and columns as close in norm as possible. Both steps are optional.

Balancing may reduce the 1-norm of the matrices, and improve the accuracy of the computed eigenvalues and/or eigenvectors in the generalized eigenvalue problem A*x = lambda*B*x.

Syntax

Fortran specification:

```
use armpl_library

subroutine zggbal(JOB, N, A, LDA, B, LDB, ILO, IHI, LSCALE, RSCALE, WORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void zggbal_(const char *job, const armpl_int_t *n, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, armpl_int_t *ilo, armpl_int_t *ihi,
             double *lscale, double *rscale, double *work, armpl_int_t *info,
             ... );
```

Parameters

JOB Input parameter.

JOB is CHARACTER*1

Specifies the operations to be performed on A and B: = 'N': none: simply set ILO = 1, IHI = N, LSCALE(I) = 1.0 and RSCALE(I) = 1.0 for i=1,...,N; = 'P': permute only; = 'S': scale only; = 'B': both permute and scale.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the input matrix A. On exit, A is overwritten by the balanced matrix. If JOB = 'N', A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). On entry, the input matrix B. On exit, B is overwritten by the balanced matrix. If JOB = 'N', B is not referenced.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

ILO Output parameter.

ILO is INTEGER

IHI Output parameter.

IHI is INTEGER

ILO and IHI are set to integers such that on exit A(i,j) = 0 and B(i,j) = 0 if i > j and j = 1,...,ILO-1 or i = IHI+1,...,N. If JOB = 'N' or 'S', ILO = 1 and IHI = N.

LSCALE Output parameter.

LSCALE is DOUBLE PRECISION

LSCALE is an array, dimension (N). Details of the permutations and scaling factors applied to the left side of A and B. If P(j) is the index of the row interchanged with row j, and D(j) is the scaling factor applied to row j, then LSCALE(j) = P(j) for J = 1,...,ILO-1 = D(j) for J = ILO,...,IHI = P(j) for J = IHI+1,...,N. The order in which the interchanges are made is N to IHI+1, then 1 to ILO-1.

RSCALE Output parameter.

RSCALE is DOUBLE PRECISION

RSCALE is an array, dimension (N). Details of the permutations and scaling factors applied to the right side of A and B. If P(j) is the index of the column interchanged with column j, and D(j) is the scaling factor applied to column j, then RSCALE(j) = P(j) for J = 1,...,ILO-1 = D(j) for J = ILO,...,IHI = P(j) for J = IHI+1,...,N. The order in which the interchanges are made is N to IHI+1, then 1 to ILO-1.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (lwork). lwork must be at least max(1,6*N) when JOB = 'S' or 'B', and at least 1 when JOB = 'N' or 'P'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cggbal*, *dggbal* and *sggbal*. It also exists with a native C interface as *LAPACKE_zggbal*.

4.14.51 zgges

zgges computes for a pair of N-by-N complex nonsymmetric matrices (A,B), the generalized eigenvalues, the generalized complex Schur form (S, T), and optionally left and/or right Schur vectors (VSL and VSR). This gives the generalized Schur factorization

```
(A,B) = ( (VSL)*S*(VSR)**H, (VSL)*T*(VSR)**H )
```

where $(VSR)^H$ is the conjugate-transpose of VSR.

Optionally, it also orders the eigenvalues so that a selected cluster of eigenvalues appears in the leading diagonal blocks of the upper triangular matrix S and the upper triangular matrix T. The leading columns of VSL and VSR then form an unitary basis for the corresponding left and right eigenspaces (deflating subspaces).

(If only the generalized eigenvalues are needed, use the driver ZGGEV instead, which is faster.)

A generalized eigenvalue for a pair of matrices (A,B) is a scalar w or a ratio alpha/beta = w, such that A - w*B is singular. It is usually represented as the pair (alpha,beta), as there is a reasonable interpretation for beta=0, and even for both being zero.

A pair of matrices (S,T) is in generalized complex Schur form if S and T are upper triangular and, in addition, the diagonal elements of T are non-negative real numbers.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgges(JOBVSL, JOBVSR, SORT, SELCTG, N, A, LDA, B, LDB, SDIM, ALPHA,
                 BETA, VSL, LDVSL, VSR, LDVSR, WORK, LWORK, RWORK, BWORK,
                 INFO)
```

C specification:

```
#include "armpl.h"

void zgges_(const char *jobvsl, const char *jobvsr, const char *sort,
            ARMPL_ZGGES_DELCTG delctg, const armpl_int_t *n,
            armpl_doublecomplex_t *a, const armpl_int_t *lda,
            armpl_doublecomplex_t *b, const armpl_int_t *ldb,
            armpl_int_t *sdim, armpl_doublecomplex_t *alpha,
            armpl_doublecomplex_t *beta, armpl_doublecomplex_t *vsl,
            const armpl_int_t *ldvsl, armpl_doublecomplex_t *vsr,
            const armpl_int_t *ldvsr, armpl_doublecomplex_t *work,
            const armpl_int_t *lwork, double *rwork, armpl_int_t *bwork,
            armpl_int_t *info, ... );
```

Parameters

JOBVSL Input parameter.

JOBVSL is CHARACTER*1

= 'N': do not compute the left Schur vectors; = 'V': compute the left Schur vectors.

JOBVSR Input parameter.

JOBVSR is CHARACTER*1

= 'N': do not compute the right Schur vectors; = 'V': compute the right Schur vectors.

SORT Input parameter.

SORT is CHARACTER*1

Specifies whether or not to order the eigenvalues on the diagonal of the generalized Schur form. = 'N': Eigenvalues are not ordered; = 'S': Eigenvalues are ordered (see SELCTG).

SELCTG Input parameter.

SELCTG is a LOGICAL FUNCTION of two COMPLEX*16 arguments

SELCTG must be declared EXTERNAL in the calling subroutine. If SORT = 'N', SELCTG is not referenced. If SORT = 'S', SELCTG is used to select eigenvalues to sort to the top left of the Schur form. An eigenvalue ALPHA(j)/BETA(j) is selected if SELCTG(ALPHA(j),BETA(j)) is true.

Note that a selected complex eigenvalue may no longer satisfy SELCTG(ALPHA(j),BETA(j)) = .TRUE. after ordering, since ordering may change the value of complex eigenvalues (especially if the eigenvalue is ill-conditioned), in this case INFO is set to N+2 (See INFO below).

N Input parameter.

N is INTEGER

The order of the matrices A, B, VSL, and VSR. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the first of the pair of matrices. On exit, A has been overwritten by its generalized Schur form S.

LDA Input parameter.

LDA is INTEGER

The leading dimension of A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). On entry, the second of the pair of matrices. On exit, B has been overwritten by its generalized Schur form T.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B. LDB >= max(1, N).

SDIM Output parameter.

SDIM is INTEGER

If SORT = 'N', SDIM = 0. If SORT = 'S', SDIM = number of eigenvalues (after sorting) for which SELCTG is true.

ALPHA Output parameter.

ALPHA is COMPLEX*16

ALPHA is an array, dimension (N) .

BETA Output parameter.

BETA is COMPLEX*16

BETA is an array, dimension (N). On exit, ALPHA(j)/BETA(j), j=1,..., N, will be the generalized eigenvalues. ALPHA(j), j=1,...,N and BETA(j), j=1,...,N are the diagonals of the complex Schur form (A, B) output by ZGGES. The BETA(j) will be non-negative real.

Note: the quotients ALPHA(j)/BETA(j) may easily over- or underflow, and BETA(j) may even be zero. Thus, the user should avoid naively computing the ratio alpha/beta. However, ALPHA will be always less than and usually comparable with norm(A) in magnitude, and BETA always less than and usually comparable with norm(B).

VSL Output parameter.

VSL is COMPLEX*16

VSL is an array, dimension (LDVSL, N). If JOBVSL = 'V', VSL will contain the left Schur vectors. Not referenced if JOBVSL = 'N'.

LDVSL Input parameter.

LDVSL is INTEGER

The leading dimension of the matrix VSL. LDVSL >= 1, and if JOBVSL = 'V', LDVSL >= N.

VSR Output parameter.

VSR is COMPLEX*16

VSR is an array, dimension (LDVSR, N). If JOBVSR = 'V', VSR will contain the right Schur vectors. Not referenced if JOBVSR = 'N'.

LDVSR Input parameter.

LDVSR is INTEGER

The leading dimension of the matrix VSR. LDVSR >= 1, and if JOBVSR = 'V', LDVSR >= N.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,2*N). For good performance, LWORK must generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (8*N) .

BWORK Output parameter.

BWORK is LOGICAL

BWORK is an array, dimension (N). Not referenced if SORT = 'N'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. =1,...,N: The QZ iteration failed. (A, B) are not in Schur form, but ALPHA(j) and BETA(j) should be correct for j=INFO+1,...,N. > N: =N+1: other than QZ iteration failed in ZHGEQZ =N+2: after reordering, roundoff changed values of some complex eigenvalues so that leading eigenvalues in the Generalized Schur form no longer satisfy SELCTG=.TRUE. This could also be caused due to scaling. =N+3: reordering failed in ZTGSEN.

Related Information

For this routine in other precisions, please see *cgges*, *dgges* and *sgges*. It also exists with a native C interface as *LAPACKE_zgges*.

4.14.52 zgges3

zgges3 computes for a pair of N-by-N complex nonsymmetric matrices (A,B), the generalized eigenvalues, the generalized complex Schur form (S, T), and optionally left and/or right Schur vectors (VSL and VSR). This gives the generalized Schur factorization

```
(A,B) = ( (VSL)*S*(VSR)**H, (VSL)*T*(VSR)**H )
```

where $(VSR)^H$ is the conjugate-transpose of VSR.

Optionally, it also orders the eigenvalues so that a selected cluster of eigenvalues appears in the leading diagonal blocks of the upper triangular matrix S and the upper triangular matrix T. The leading columns of VSL and VSR then form an unitary basis for the corresponding left and right eigenspaces (deflating subspaces).

(If only the generalized eigenvalues are needed, use the driver ZGGEV instead, which is faster.)

A generalized eigenvalue for a pair of matrices (A,B) is a scalar w or a ratio alpha/beta = w, such that A - w*B is singular. It is usually represented as the pair (alpha,beta), as there is a reasonable interpretation for beta=0, and even for both being zero.

A pair of matrices (S,T) is in generalized complex Schur form if S and T are upper triangular and, in addition, the diagonal elements of T are non-negative real numbers.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgges3(JOBVSL, JOBVSR, SORT, SELCTG, N, A, LDA, B, LDB, SDIM,
                  ALPHA, BETA, VSL, LDVSL, VSR, LDVSR, WORK, LWORK, RWORK,
                  BWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgges3_(const char *jobvsl, const char *jobvsr, const char *sort,
             ARMPL_ZGGES3_SELCTG selctg, const armpl_int_t *n,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_doublecomplex_t *b, const armpl_int_t *ldb,
             armpl_int_t *sdim, armpl_doublecomplex_t *alpha,
             armpl_doublecomplex_t *beta, armpl_doublecomplex_t *vsl,
             const armpl_int_t *ldvsl, armpl_doublecomplex_t *vsr,
             const armpl_int_t *ldvsr, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, double *rwork, armpl_int_t *bwork,
             armpl_int_t *info, ... );
```

Parameters

JOBVSL Input parameter.

JOBVSL is CHARACTER*1

= 'N': do not compute the left Schur vectors; = 'V': compute the left Schur vectors.

JOBVSR Input parameter.

JOBVSR is CHARACTER*1

= 'N': do not compute the right Schur vectors; = 'V': compute the right Schur vectors.

SORT Input parameter.

SORT is CHARACTER*1

Specifies whether or not to order the eigenvalues on the diagonal of the generalized Schur form. = 'N': Eigenvalues are not ordered; = 'S': Eigenvalues are ordered (see SELCTG).

SELCTG Input parameter.

SELCTG is a LOGICAL FUNCTION of two COMPLEX*16 arguments

SELCTG must be declared EXTERNAL in the calling subroutine. If SORT = 'N', SELCTG is not referenced. If SORT = 'S', SELCTG is used to select eigenvalues to sort to the top left of the Schur form. An eigenvalue ALPHA(j)/BETA(j) is selected if SELCTG(ALPHA(j),BETA(j)) is true.

Note that a selected complex eigenvalue may no longer satisfy SELCTG(ALPHA(j),BETA(j)) = .TRUE. after ordering, since ordering may change the value of complex eigenvalues (especially if the eigenvalue is ill-conditioned), in this case INFO is set to N+2 (See INFO below).

N Input parameter.

N is INTEGER

The order of the matrices A, B, VSL, and VSR. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the first of the pair of matrices. On exit, A has been overwritten by its generalized Schur form S.

LDA Input parameter.

LDA is INTEGER

The leading dimension of A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). On entry, the second of the pair of matrices. On exit, B has been overwritten by its generalized Schur form T.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B. LDB >= max(1, N).

SDIM Output parameter.

SDIM is INTEGER

If SORT = 'N', SDIM = 0. If SORT = 'S', SDIM = number of eigenvalues (after sorting) for which SELCTG is true.

ALPHA Output parameter.

ALPHA is COMPLEX*16

ALPHA is an array, dimension (N) .

BETA Output parameter.

BETA is COMPLEX*16

BETA is an array, dimension (N). On exit, ALPHA(j)/BETA(j), j=1,..., N, will be the generalized eigenvalues. ALPHA(j), j=1,...,N and BETA(j), j=1,...,N are the diagonals of the complex Schur form (A, B) output by ZGGES3. The BETA(j) will be non-negative real.

Note: the quotients ALPHA(j)/BETA(j) may easily over- or underflow, and BETA(j) may even be zero. Thus, the user should avoid naively computing the ratio alpha/beta. However, ALPHA will be always less than and usually comparable with norm(A) in magnitude, and BETA always less than and usually comparable with norm(B).

VSL Output parameter.

VSL is COMPLEX*16

VSL is an array, dimension (LDVSL, N). If JOBVSL = 'V', VSL will contain the left Schur vectors. Not referenced if JOBVSL = 'N'.

LDVSL Input parameter.

LDVSL is INTEGER

The leading dimension of the matrix VSL. LDVSL >= 1, and if JOBVSL = 'V', LDVSL >= N.

VSR Output parameter.

VSR is COMPLEX*16

VSR is an array, dimension (LDVSR, N). If JOBVSR = 'V', VSR will contain the right Schur vectors. Not referenced if JOBVSR = 'N'.

LDVSR Input parameter.

LDVSR is INTEGER

The leading dimension of the matrix VSR. LDVSR >= 1, and if JOBVSR = 'V', LDVSR >= N.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (8*N) .

BWORK Output parameter.

BWORK is LOGICAL

BWORK is an array, dimension (N). Not referenced if SORT = 'N'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. =1,...,N: The QZ iteration failed. (A, B) are not in Schur form, but ALPHA(j) and BETA(j) should be correct for j=INFO+1,...,N. > N: =N+1: other than QZ iteration failed in ZHGEQZ =N+2: after reordering, roundoff changed values of some complex eigenvalues so that leading eigenvalues in the Generalized Schur form no longer satisfy SELCTG=.TRUE. This could also be caused due to scaling. =N+3: reordering failed in ZTGSEN.

Related Information

For this routine in other precisions, please see *cgges3*, *dgges3* and *sgges3*. It also exists with a native C interface as *LAPACKE_zgges3*.

4.14.53 zggesx

zggesx computes for a pair of N-by-N complex nonsymmetric matrices (A,B), the generalized eigenvalues, the complex Schur form (S,T), and, optionally, the left and/or right matrices of Schur vectors (VSL and VSR). This gives the generalized Schur factorization

```
(A,B) = ( (VSL) S (VSR)**H, (VSL) T (VSR)**H )
```

where $(VSR)^H$ is the conjugate-transpose of VSR.

Optionally, it also orders the eigenvalues so that a selected cluster of eigenvalues appears in the leading diagonal blocks of the upper triangular matrix S and the upper triangular matrix T; computes a reciprocal condition number for the average of the selected eigenvalues (RCONDE); and computes a reciprocal condition number for the right and left deflating subspaces corresponding to the selected eigenvalues (RCONDV). The leading columns of VSL and VSR then form an orthonormal basis for the corresponding left and right eigenspaces (deflating subspaces).

A generalized eigenvalue for a pair of matrices (A,B) is a scalar w or a ratio alpha/beta = w, such that A - w*B is singular. It is usually represented as the pair (alpha,beta), as there is a reasonable interpretation for beta=0 or for both being zero.

A pair of matrices (S,T) is in generalized complex Schur form if T is upper triangular with non-negative diagonal and S is upper triangular.

Syntax

Fortran specification:

```
use armpl_library

subroutine zggesx(JOBVSL, JOBVSR, SORT, SELCTG, SENSE, N, A, LDA, B, LDB,
                  SDIM, ALPHA, BETA, VSL, LDVSL, VSR, LDVSR, RCONDE, RCONDV,
                  WORK, LWORK, RWORK, IWORK, LIWORK, BWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zggesx_(const char *jobvsl, const char *jobvsr, const char *sort,
             ARMPL_ZGGESX_DELCTG delctg, const char *sense,
             const armpl_int_t *n, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, armpl_int_t *sdim,
             armpl_doublecomplex_t *alpha, armpl_doublecomplex_t *beta,
             armpl_doublecomplex_t *vsl, const armpl_int_t *ldvsl,
             armpl_doublecomplex_t *vsr, const armpl_int_t *ldvsr,
             double *rconde, double *rcondv, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, double *rwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *bwork, armpl_int_t *info,
             ... );
```

Parameters

JOBVSL Input parameter.

JOBVSL is CHARACTER*1

= 'N': do not compute the left Schur vectors; = 'V': compute the left Schur vectors.

JOBVSR Input parameter.

JOBVSR is CHARACTER*1

= 'N': do not compute the right Schur vectors; = 'V': compute the right Schur vectors.

SORT Input parameter.

SORT is CHARACTER*1

Specifies whether or not to order the eigenvalues on the diagonal of the generalized Schur form. = 'N': Eigenvalues are not ordered; = 'S': Eigenvalues are ordered (see SELCTG).

SELCTG Input parameter.

SELCTG is a LOGICAL FUNCTION of two COMPLEX*16 arguments

SELCTG must be declared EXTERNAL in the calling subroutine. If SORT = 'N', SELCTG is not referenced. If SORT = 'S', SELCTG is used to select eigenvalues to sort to the top left of the Schur form. Note that a selected complex eigenvalue may no longer satisfy SELCTG(ALPHA(j),BETA(j)) = .TRUE. after ordering, since ordering may change the value of complex eigenvalues (especially if the eigenvalue is ill-conditioned), in this case INFO is set to N+3 see INFO below).

SENSE Input parameter.

SENSE is CHARACTER*1

Determines which reciprocal condition numbers are computed. = 'N' : None are computed; = 'E' : Computed for average of selected eigenvalues only; = 'V' : Computed for selected deflating subspaces only; = 'B' : Computed for both. If SENSE = 'E', 'V', or 'B', SORT must equal 'S'.

N Input parameter.

N is INTEGER

The order of the matrices A, B, VSL, and VSR. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the first of the pair of matrices. On exit, A has been overwritten by its generalized Schur form S.

LDA Input parameter.

LDA is INTEGER

The leading dimension of A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). On entry, the second of the pair of matrices. On exit, B has been overwritten by its generalized Schur form T.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B. LDB >= max(1, N).

SDIM Output parameter.

SDIM is INTEGER

If SORT = 'N', SDIM = 0. If SORT = 'S', SDIM = number of eigenvalues (after sorting) for which SELCTG is true.

ALPHA Output parameter.

ALPHA is COMPLEX*16

ALPHA is an array, dimension (N) .

BETA Output parameter.

BETA is COMPLEX*16

BETA is an array, dimension (N). On exit, ALPHA(j)/BETA(j), j=1,..., N, will be the generalized eigenvalues. ALPHA(j) and BETA(j),j=1,...,N are the diagonals of the complex Schur form (S,T). BETA(j) will be non-negative real.

Note: the quotients ALPHA(j)/BETA(j) may easily over- or underflow, and BETA(j) may even be zero. Thus, the user should avoid naively computing the ratio alpha/beta. However, ALPHA will be always less than and usually comparable with norm(A) in magnitude, and BETA always less than and usually comparable with norm(B).

VSL Output parameter.

VSL is COMPLEX*16

VSL is an array, dimension (LDVSL, N). If JOBVSL = 'V', VSL will contain the left Schur vectors. Not referenced if JOBVSL = 'N'.

LDVSL Input parameter.

LDVSL is INTEGER

The leading dimension of the matrix VSL. LDVSL >=1, and if JOBVSL = 'V', LDVSL >= N.

VSR Output parameter.

VSR is COMPLEX*16

VSR is an array, dimension (LDVSR, N). If JOBVSR = 'V', VSR will contain the right Schur vectors. Not referenced if JOBVSR = 'N'.

LDVSR Input parameter.

LDVSR is INTEGER

The leading dimension of the matrix VSR. LDVSR >= 1, and if JOBVSR = 'V', LDVSR >= N.

RCONDE Output parameter.

RCONDE is DOUBLE PRECISION

RCONDE is an array, dimension (2). If SENSE = 'E' or 'B', RCONDE(1) and RCONDE(2) contain the reciprocal condition numbers for the average of the selected eigenvalues. Not referenced if SENSE = 'N' or 'V'.

RCONDV Output parameter.

RCONDV is DOUBLE PRECISION

RCONDV is an array, dimension (2). If SENSE = 'V' or 'B', RCONDV(1) and RCONDV(2) contain the reciprocal condition number for the selected deflating subspaces. Not referenced if SENSE = 'N' or 'E'.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If N = 0, LWORK >= 1, else if SENSE = 'E', 'V', or 'B', LWORK >= MAX(1,2*N,2*SDIM*(N-SDIM)), else LWORK >= MAX(1,2*N). Note that 2*SDIM*(N-SDIM) <= N*N/2. Note also that an error is only returned if LWORK < MAX(1,2*N), but if SENSE = 'E' or 'V' or 'B' this may not be large enough.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the bound on the optimal size of the WORK array and the minimum size of the IWORK array, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (8* N). Real workspace.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the minimum LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. If SENSE = 'N' or N = 0, LIWORK >= 1, otherwise LIWORK >= N+2.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the bound on the optimal size of the WORK array and the minimum size of the IWORK array, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

BWORK Output parameter.

BWORK is LOGICAL

BWORK is an array, dimension (N). Not referenced if SORT = 'N'.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. = 1,...,N: The QZ iteration failed. (A, B) are not in Schur form, but ALPHA(j) and BETA(j) should be correct for j=INFO+1,...,N. > N: =N+1: other than QZ iteration failed in ZHGEQZ =N+2: after reordering, roundoff changed values of some complex eigenvalues so that leading eigenvalues in the Generalized Schur form no longer satisfy SELCTG=.TRUE. This could also be caused due to scaling. =N+3: reordering failed in ZTGSEN.

Related Information

For this routine in other precisions, please see *cggesx*, *dggesx* and *sggesx*. It also exists with a native C interface as *LAPACKE_zggesx*.

4.14.54 zggev

zggev computes for a pair of N-by-N complex nonsymmetric matrices (A,B), the generalized eigenvalues, and optionally, the left and/or right generalized eigenvectors.

A generalized eigenvalue for a pair of matrices (A,B) is a scalar lambda or a ratio alpha/beta = lambda, such that A - lambda*B is singular. It is usually represented as the pair (alpha,beta), as there is a reasonable interpretation for beta=0, and even for both being zero.

The right generalized eigenvector v(j) corresponding to the generalized eigenvalue lambda(j) of (A,B) satisfies

```
A * v(j) = lambda(j) * B * v(j).
```

The left generalized eigenvector u(j) corresponding to the generalized eigenvalues lambda(j) of (A,B) satisfies

```
u(j)**H * A = lambda(j) * u(j)**H * B
```

where $u(j)^H$ is the conjugate-transpose of u(j).

Syntax

Fortran specification:

```
use armpl_library

subroutine zggev(JOBVL, JOBVR, N, A, LDA, B, LDB, ALPHA, BETA, VL, LDVL, VR,
                 LDVR, WORK, LWORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zggev_(const char *jobvl, const char *jobvr, const armpl_int_t *n,
            armpl_doublecomplex_t *a, const armpl_int_t *lda,
            armpl_doublecomplex_t *b, const armpl_int_t *ldb,
            armpl_doublecomplex_t *alpha, armpl_doublecomplex_t *beta,
```

(continues on next page)

(continued from previous page)

```
            armpl_doublecomplex_t *vl, const armpl_int_t *ldvl,
            armpl_doublecomplex_t *vr, const armpl_int_t *ldvr,
            armpl_doublecomplex_t *work, const armpl_int_t *lwork,
            double *rwork, armpl_int_t *info, ... );
```

Parameters

JOBVL Input parameter.

JOBVL is CHARACTER*1

= 'N': do not compute the left generalized eigenvectors; = 'V': compute the left generalized eigenvectors.

JOBVR Input parameter.

JOBVR is CHARACTER*1

= 'N': do not compute the right generalized eigenvectors; = 'V': compute the right generalized eigenvectors.

N Input parameter.

N is INTEGER

The order of the matrices A, B, VL, and VR. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the matrix A in the pair (A, B). On exit, A has been overwritten.

LDA Input parameter.

LDA is INTEGER

The leading dimension of A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). On entry, the matrix B in the pair (A, B). On exit, B has been overwritten.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B. LDB >= max(1, N).

ALPHA Output parameter.

ALPHA is COMPLEX*16

ALPHA is an array, dimension (N) .

BETA Output parameter.

BETA is COMPLEX*16

BETA is an array, dimension (N). On exit, ALPHA(j)/BETA(j), j=1,..., N, will be the generalized eigenvalues.

Note: the quotients ALPHA(j)/BETA(j) may easily over- or underflow, and BETA(j) may even be zero. Thus, the user should avoid naively computing the ratio alpha/beta. However, ALPHA will be always less than and usually comparable with norm(A) in magnitude, and BETA always less than and usually comparable with norm(B).

VL Output parameter.

VL is COMPLEX*16

VL is an array, dimension (LDVL, N). If JOBVL = 'V', the left generalized eigenvectors u(j) are stored one after another in the columns of VL, in the same order as their eigenvalues. Each eigenvector is scaled so the largest component has abs(real part) + abs(imag. part) = 1. Not referenced if JOBVL = 'N'.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the matrix VL. LDVL >= 1, and if JOBVL = 'V', LDVL >= N.

VR Output parameter.

VR is COMPLEX*16

VR is an array, dimension (LDVR, N). If JOBVR = 'V', the right generalized eigenvectors v(j) are stored one after another in the columns of VR, in the same order as their eigenvalues. Each eigenvector is scaled so the largest component has abs(real part) + abs(imag. part) = 1. Not referenced if JOBVR = 'N'.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the matrix VR. LDVR >= 1, and if JOBVR = 'V', LDVR >= N.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,2*N). For good performance, LWORK must generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (8*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. =1,. . . ,N: The QZ iteration failed. No eigenvectors have been calculated, but ALPHA(j) and BETA(j) should be correct for j=INFO+1,. . . ,N. > N: =N+1: other then QZ iteration failed in DHGEQZ, =N+2: error return from DTGEVC.

Related Information

For this routine in other precisions, please see *cggev*, *dggev* and *sggev*. It also exists with a native C interface as *LAPACKE_zggev*.

4.14.55 zggev3

zggev3 computes for a pair of N-by-N complex nonsymmetric matrices (A,B), the generalized eigenvalues, and optionally, the left and/or right generalized eigenvectors.

A generalized eigenvalue for a pair of matrices (A,B) is a scalar lambda or a ratio alpha/beta = lambda, such that A - lambda*B is singular. It is usually represented as the pair (alpha,beta), as there is a reasonable interpretation for beta=0, and even for both being zero.

The right generalized eigenvector v(j) corresponding to the generalized eigenvalue lambda(j) of (A,B) satisfies

```
A * v(j) = lambda(j) * B * v(j).
```

The left generalized eigenvector u(j) corresponding to the generalized eigenvalues lambda(j) of (A,B) satisfies

```
u(j)**H * A = lambda(j) * u(j)**H * B
```

where $u(j)^H$ is the conjugate-transpose of u(j).

Syntax

Fortran specification:

```
use armpl_library

subroutine zggev3(JOBVL, JOBVR, N, A, LDA, B, LDB, ALPHA, BETA, VL, LDVL, VR,
                  LDVR, WORK, LWORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zggev3_(const char *jobvl, const char *jobvr, const armpl_int_t *n,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_doublecomplex_t *b, const armpl_int_t *ldb,
             armpl_doublecomplex_t *alpha, armpl_doublecomplex_t *beta,
             armpl_doublecomplex_t *vl, const armpl_int_t *ldvl,
             armpl_doublecomplex_t *vr, const armpl_int_t *ldvr,
             armpl_doublecomplex_t *work, const armpl_int_t *lwork,
             double *rwork, armpl_int_t *info, ... );
```

Parameters

JOBVL Input parameter.

JOBVL is CHARACTER*1

= 'N': do not compute the left generalized eigenvectors; = 'V': compute the left generalized eigenvectors.

JOBVR Input parameter.

JOBVR is CHARACTER*1

= 'N': do not compute the right generalized eigenvectors; = 'V': compute the right generalized eigenvectors.

N Input parameter.

N is INTEGER

The order of the matrices A, B, VL, and VR. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the matrix A in the pair (A, B). On exit, A has been overwritten.

LDA Input parameter.

LDA is INTEGER

The leading dimension of A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). On entry, the matrix B in the pair (A, B). On exit, B has been overwritten.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B. LDB >= max(1, N).

ALPHA Output parameter.

ALPHA is COMPLEX*16

ALPHA is an array, dimension (N) .

BETA Output parameter.

BETA is COMPLEX*16

BETA is an array, dimension (N). On exit, ALPHA(j)/BETA(j), j=1,..., N, will be the generalized eigenvalues.

Note: the quotients ALPHA(j)/BETA(j) may easily over- or underflow, and BETA(j) may even be zero. Thus, the user should avoid naively computing the ratio alpha/beta. However, ALPHA will be always less than and usually comparable with norm(A) in magnitude, and BETA always less than and usually comparable with norm(B).

VL Output parameter.

VL is COMPLEX*16

VL is an array, dimension (LDVL, N). If JOBVL = 'V', the left generalized eigenvectors u(j) are stored one after another in the columns of VL, in the same order as their eigenvalues. Each eigenvector is scaled so the largest component has abs(real part) + abs(imag. part) = 1. Not referenced if JOBVL = 'N'.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the matrix VL. LDVL >= 1, and if JOBVL = 'V', LDVL >= N.

VR Output parameter.

VR is COMPLEX*16

VR is an array, dimension (LDVR, N). If JOBVR = 'V', the right generalized eigenvectors v(j) are stored one after another in the columns of VR, in the same order as their eigenvalues. Each eigenvector is scaled so the largest component has abs(real part) + abs(imag. part) = 1. Not referenced if JOBVR = 'N'.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the matrix VR. LDVR >= 1, and if JOBVR = 'V', LDVR >= N.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (8*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. =1,. . . ,N: The QZ iteration failed. No eigenvectors have been calculated, but ALPHA(j) and BETA(j) should be correct for j=INFO+1,. . . ,N. > N: =N+1: other then QZ iteration failed in DHGEQZ, =N+2: error return from DTGEVC.

Related Information

For this routine in other precisions, please see *cggev3*, *dggev3* and *sggev3*. It also exists with a native C interface as *LAPACKE_zggev3*.

4.14.56 zggevx

zggevx computes for a pair of N-by-N complex nonsymmetric matrices (A,B) the generalized eigenvalues, and optionally, the left and/or right generalized eigenvectors.

Optionally, it also computes a balancing transformation to improve the conditioning of the eigenvalues and eigenvectors (ILO, IHI, LSCALE, RSCALE, ABNRM, and BBNRM), reciprocal condition numbers for the eigenvalues (RCONDE), and reciprocal condition numbers for the right eigenvectors (RCONDV).

A generalized eigenvalue for a pair of matrices (A,B) is a scalar lambda or a ratio alpha/beta = lambda, such that A - lambda*B is singular. It is usually represented as the pair (alpha,beta), as there is a reasonable interpretation for beta=0, and even for both being zero.

The right eigenvector v(j) corresponding to the eigenvalue lambda(j) of (A,B) satisfies

```
A * v(j) = lambda(j) * B * v(j) .
```

The left eigenvector u(j) corresponding to the eigenvalue lambda(j) of (A,B) satisfies

```
u(j)**H * A  = lambda(j) * u(j)**H * B.
```

where $u(j)^H$ is the conjugate-transpose of u(j).

Syntax

Fortran specification:

```
use armpl_library

subroutine zggevx(BALANC, JOBVL, JOBVR, SENSE, N, A, LDA, B, LDB, ALPHA, BETA,
                  VL, LDVL, VR, LDVR, ILO, IHI, LSCALE, RSCALE, ABNRM, BBNRM,
                  RCONDE, RCONDV, WORK, LWORK, RWORK, IWORK, BWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zggevx_(const char *balanc, const char *jobvl, const char *jobvr,
             const char *sense, const armpl_int_t *n,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_doublecomplex_t *b, const armpl_int_t *ldb,
             armpl_doublecomplex_t *alpha, armpl_doublecomplex_t *beta,
             armpl_doublecomplex_t *vl, const armpl_int_t *ldvl,
             armpl_doublecomplex_t *vr, const armpl_int_t *ldvr,
             armpl_int_t *ilo, armpl_int_t *ihi, double *lscale,
             double *rscale, double *abnrm, double *bbnrm, double *rconde,
             double *rcondv, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, double *rwork, armpl_int_t *iwork,
             armpl_int_t *bwork, armpl_int_t *info, ... );
```

Parameters

BALANC Input parameter.

BALANC is CHARACTER*1

Specifies the balance option to be performed: = 'N': do not diagonally scale or permute; = 'P': permute only; = 'S': scale only; = 'B': both permute and scale. Computed reciprocal condition numbers will be for the matrices after permuting and/or balancing. Permuting does not change condition numbers (in exact arithmetic), but balancing does.

JOBVL Input parameter.

JOBVL is CHARACTER*1

= 'N': do not compute the left generalized eigenvectors; = 'V': compute the left generalized eigenvectors.

JOBVR Input parameter.

JOBVR is CHARACTER*1

= 'N': do not compute the right generalized eigenvectors; = 'V': compute the right generalized eigenvectors.

SENSE Input parameter.

SENSE is CHARACTER*1

Determines which reciprocal condition numbers are computed. = 'N': none are computed; = 'E': computed for eigenvalues only; = 'V': computed for eigenvectors only; = 'B': computed for eigenvalues and eigenvectors.

N Input parameter.

N is INTEGER

The order of the matrices A, B, VL, and VR. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the matrix A in the pair (A, B). On exit, A has been overwritten. If JOBVL='V' or JOBVR='V' or both, then A contains the first part of the complex Schur form of the "balanced" versions of the input A and B.

LDA Input parameter.

LDA is INTEGER

The leading dimension of A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). On entry, the matrix B in the pair (A, B). On exit, B has been overwritten. If JOBVL='V' or JOBVR='V' or both, then B contains the second part of the complex Schur form of the "balanced" versions of the input A and B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B. LDB >= max(1, N).

ALPHA Output parameter.

ALPHA is COMPLEX*16

ALPHA is an array, dimension (N) .

BETA Output parameter.

BETA is COMPLEX*16

BETA is an array, dimension (N). On exit, ALPHA(j)/BETA(j), j=1,..., N, will be the generalized eigenvalues.

Note: the quotient ALPHA(j)/BETA(j)) may easily over- or underflow, and BETA(j) may even be zero. Thus, the user should avoid naively computing the ratio ALPHA/BETA. However, ALPHA will be always less than and usually comparable with norm(A) in magnitude, and BETA always less than and usually comparable with norm(B).

VL Output parameter.

VL is COMPLEX*16

VL is an array, dimension (LDVL, N). If JOBVL = 'V', the left generalized eigenvectors u(j) are stored one after another in the columns of VL, in the same order as their eigenvalues. Each eigenvector will be scaled so the largest component will have abs(real part) + abs(imag. part) = 1. Not referenced if JOBVL = 'N'.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the matrix VL. LDVL >= 1, and if JOBVL = 'V', LDVL >= N.

VR Output parameter.

VR is COMPLEX*16

VR is an array, dimension (LDVR, N). If JOBVR = 'V', the right generalized eigenvectors v(j) are stored one after another in the columns of VR, in the same order as their eigenvalues. Each eigenvector will be scaled so the largest component will have abs(real part) + abs(imag. part) = 1. Not referenced if JOBVR = 'N'.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the matrix VR. LDVR >= 1, and if JOBVR = 'V', LDVR >= N.

ILO Output parameter.

ILO is INTEGER

IHI Output parameter.

IHI is INTEGER

ILO and IHI are integer values such that on exit A(i,j) = 0 and B(i,j) = 0 if i > j and j = 1,...,ILO-1 or i = IHI+1,...,N. If BALANC = 'N' or 'S', ILO = 1 and IHI = N.

LSCALE Output parameter.

LSCALE is DOUBLE PRECISION

LSCALE is an array, dimension (N). Details of the permutations and scaling factors applied to the left side of A and B. If PL(j) is the index of the row interchanged with row j, and DL(j) is the scaling factor applied to

row j, then LSCALE(j) = PL(j) for j = 1,...,ILO-1 = DL(j) for j = ILO,...,IHI = PL(j) for j = IHI+1,...,N. The order in which the interchanges are made is N to IHI+1, then 1 to ILO-1.

RSCALE Output parameter.

RSCALE is DOUBLE PRECISION

RSCALE is an array, dimension (N). Details of the permutations and scaling factors applied to the right side of A and B. If PR(j) is the index of the column interchanged with column j, and DR(j) is the scaling factor applied to column j, then RSCALE(j) = PR(j) for j = 1,...,ILO-1 = DR(j) for j = ILO,...,IHI = PR(j) for j = IHI+1,...,N The order in which the interchanges are made is N to IHI+1, then 1 to ILO-1.

ABNRM Output parameter.

ABNRM is DOUBLE PRECISION

The one-norm of the balanced matrix A.

BBNRM Output parameter.

BBNRM is DOUBLE PRECISION

The one-norm of the balanced matrix B.

RCONDE Output parameter.

RCONDE is DOUBLE PRECISION

RCONDE is an array, dimension (N). If SENSE = 'E' or 'B', the reciprocal condition numbers of the eigenvalues, stored in consecutive elements of the array. If SENSE = 'N' or 'V', RCONDE is not referenced.

RCONDV Output parameter.

RCONDV is DOUBLE PRECISION

RCONDV is an array, dimension (N). If JOB = 'V' or 'B', the estimated reciprocal condition numbers of the eigenvectors, stored in consecutive elements of the array. If the eigenvalues cannot be reordered to compute RCONDV(j), RCONDV(j) is set to 0; this can only occur when the true value would be very small anyway. If SENSE = 'N' or 'E', RCONDV is not referenced.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1,2*N). If SENSE = 'E', LWORK >= max(1,4*N). If SENSE = 'V' or 'B', LWORK >= max(1,2*N*N+2*N).

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (lrwork). lrwork must be at least max(1,6*N) if BALANC = 'S' or 'B', and at least max(1,2*N) otherwise. Real workspace.

IWORK Output parameter.

IWORK is INTEGER array, dimension (N+2)

If SENSE = 'E', IWORK is not referenced.

BWORK Output parameter.

BWORK is LOGICAL

BWORK is an array, dimension (N). If SENSE = 'N', BWORK is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. = 1,...,N: The QZ iteration failed. No eigenvectors have been calculated, but ALPHA(j) and BETA(j) should be correct for j=INFO+1,...,N. > N: =N+1: other than QZ iteration failed in ZHGEQZ. =N+2: error return from ZTGEVC.

Related Information

For this routine in other precisions, please see *cggevx*, *dggevx* and *sggevx*. It also exists with a native C interface as *LAPACKE_zggevx*.

4.14.57 zgghd3

zgghd3 reduces a pair of complex matrices (A,B) to generalized upper Hessenberg form using unitary transformations, where A is a general matrix and B is upper triangular. The form of the generalized eigenvalue problem is

```
A*x = lambda*B*x,
```

and B is typically made upper triangular by computing its QR factorization and moving the unitary matrix Q to the left side of the equation.

This subroutine simultaneously reduces A to a Hessenberg matrix H:

```
Q**H*A*Z = H
```

and transforms B to another upper triangular matrix T:

```
Q**H*B*Z = T
```

in order to reduce the problem to its standard form

```
H*y = lambda*T*y
```

where y = Z^H *x.

The unitary matrices Q and Z are determined as products of Givens rotations. They may either be formed explicitly, or they may be postmultiplied into input matrices Q1 and Z1, so that

```
 Q1 * A * Z1**H = (Q1*Q) * H * (Z1*Z)**H
Q1 * B * Z1**H = (Q1*Q) * T * (Z1*Z)**H
```

If Q1 is the unitary matrix from the QR factorization of B in the original equation A*x = lambda*B*x, then zgghd3 reduces the original problem to generalized Hessenberg form.

This is a blocked variant of CGGHRD, using matrix-matrix multiplications for parts of the computation to enhance performance.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgghd3(COMPQ, COMPZ, N, ILO, IHI, A, LDA, B, LDB, Q, LDQ, Z, LDZ,
                  WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgghd3_(const char *compq, const char *compz, const armpl_int_t *n,
             const armpl_int_t *ilo, const armpl_int_t *ihi,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_doublecomplex_t *b, const armpl_int_t *ldb,
             armpl_doublecomplex_t *q, const armpl_int_t *ldq,
             armpl_doublecomplex_t *z, const armpl_int_t *ldz,
             armpl_doublecomplex_t *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

COMPQ Input parameter.

COMPQ is CHARACTER*1

= 'N': do not compute Q; = 'I': Q is initialized to the unit matrix, and the unitary matrix Q is returned; = 'V': Q must contain a unitary matrix Q1 on entry, and the product Q1* Q is returned.

COMPZ Input parameter.

COMPZ is CHARACTER*1

= 'N': do not compute Z; = 'I': Z is initialized to the unit matrix, and the unitary matrix Z is returned; = 'V': Z must contain a unitary matrix Z1 on entry, and the product Z1* Z is returned.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

ILO and IHI mark the rows and columns of A which are to be reduced. It is assumed that A is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N. ILO and IHI are normally set by a previous call to ZGGBAL; otherwise they should be set to 1 and N respectively. 1 <= ILO <= IHI <= N, if N > 0; ILO=1 and IHI=0, if N=0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the N-by-N general matrix to be reduced. On exit, the upper triangle and the first subdiagonal of A are overwritten with the upper Hessenberg matrix H, and the rest is set to zero.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). On entry, the N-by-N upper triangular matrix B. On exit, the upper triangular matrix T = Q^H B Z. The elements below the diagonal are set to zero.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Q Input and output parameter.

Q is COMPLEX*16

Q is an array, dimension (LDQ, N). On entry, if COMPQ = 'V', the unitary matrix Q1, typically from the QR factorization of B. On exit, if COMPQ='I', the unitary matrix Q, and if COMPQ = 'V', the product Q1*Q. Not referenced if COMPQ='N'.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= N if COMPQ='V' or 'I'; LDQ >= 1 otherwise.

Z Input and output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, N). On entry, if COMPZ = 'V', the unitary matrix Z1. On exit, if COMPZ='I', the unitary matrix Z, and if COMPZ = 'V', the product Z1*Z. Not referenced if COMPZ='N'.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= N if COMPZ='V' or 'I'; LDZ >= 1 otherwise.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (LWORK). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= 1. For optimum performance LWORK >= 6*N*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cgghd3*, *dgghd3* and *sgghd3*. It also exists with a native C interface as *LAPACKE_zgghd3*.

4.14.58 zgghrd

`zgghrd` reduces a pair of complex matrices (A,B) to generalized upper Hessenberg form using unitary transformations, where A is a general matrix and B is upper triangular. The form of the generalized eigenvalue problem is

```
A*x = lambda*B*x,
```

and B is typically made upper triangular by computing its QR factorization and moving the unitary matrix Q to the left side of the equation.

This subroutine simultaneously reduces A to a Hessenberg matrix H:

```
Q**H*A*Z = H
```

and transforms B to another upper triangular matrix T:

```
Q**H*B*Z = T
```

in order to reduce the problem to its standard form

```
H*y = lambda*T*y
```

where y = Z^H *x.

The unitary matrices Q and Z are determined as products of Givens rotations. They may either be formed explicitly, or they may be postmultiplied into input matrices Q1 and Z1, so that

```
 Q1 * A * Z1**H = (Q1*Q) * H * (Z1*Z)**H
Q1 * B * Z1**H = (Q1*Q) * T * (Z1*Z)**H
```

If Q1 is the unitary matrix from the QR factorization of B in the original equation A*x = lambda*B*x, then `zgghrd` reduces the original problem to generalized Hessenberg form.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgghrd(COMPQ, COMPZ, N, ILO, IHI, A, LDA, B, LDB, Q, LDQ, Z, LDZ,
                  INFO)
```

C specification:

```
#include "armpl.h"

void zgghrd_(const char *compq, const char *compz, const armpl_int_t *n,
             const armpl_int_t *ilo, const armpl_int_t *ihi,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_doublecomplex_t *b, const armpl_int_t *ldb,
             armpl_doublecomplex_t *q, const armpl_int_t *ldq,
             armpl_doublecomplex_t *z, const armpl_int_t *ldz,
             armpl_int_t *info, ... );
```

Parameters

COMPQ Input parameter.

`COMPQ` is `CHARACTER*1`

= 'N': do not compute Q; = 'I': Q is initialized to the unit matrix, and the unitary matrix Q is returned; = 'V': Q must contain a unitary matrix Q1 on entry, and the product Q1* Q is returned.

COMPZ Input parameter.

COMPZ is CHARACTER*1

= 'N': do not compute Z; = 'I': Z is initialized to the unit matrix, and the unitary matrix Z is returned; = 'V': Z must contain a unitary matrix Z1 on entry, and the product Z1* Z is returned.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

ILO and IHI mark the rows and columns of A which are to be reduced. It is assumed that A is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N. ILO and IHI are normally set by a previous call to ZGGBAL; otherwise they should be set to 1 and N respectively. 1 <= ILO <= IHI <= N, if N > 0; ILO=1 and IHI=0, if N=0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the N-by-N general matrix to be reduced. On exit, the upper triangle and the first subdiagonal of A are overwritten with the upper Hessenberg matrix H, and the rest is set to zero.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). On entry, the N-by-N upper triangular matrix B. On exit, the upper triangular matrix T = Q^H B Z. The elements below the diagonal are set to zero.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Q Input and output parameter.

Q is COMPLEX*16

Q is an array, dimension (LDQ, N). On entry, if COMPQ = 'V', the unitary matrix Q1, typically from the QR factorization of B. On exit, if COMPQ='I', the unitary matrix Q, and if COMPQ = 'V', the product Q1*Q. Not referenced if COMPQ='N'.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= N if COMPQ='V' or 'I'; LDQ >= 1 otherwise.

Z Input and output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, N). On entry, if COMPZ = 'V', the unitary matrix Z1. On exit, if COMPZ='I', the unitary matrix Z, and if COMPZ = 'V', the product Z1*Z. Not referenced if COMPZ='N'.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= N if COMPZ='V' or 'I'; LDZ >= 1 otherwise.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cgghrd*, *dgghrd* and *sgghrd*. It also exists with a native C interface as *LAPACKE_zgghrd*.

4.14.59 zhgeqz

zhgeqz computes the eigenvalues of a complex matrix pair (H,T), where H is an upper Hessenberg matrix and T is upper triangular, using the single-shift QZ method. Matrix pairs of this type are produced by the reduction to generalized upper Hessenberg form of a complex matrix pair (A,B):

```
A = Q1*H*Z1**H,  B = Q1*T*Z1**H,
```

as computed by ZGGHRD.

If JOB='S', then the Hessenberg-triangular pair (H,T) is also reduced to generalized Schur form,

```
H = Q*S*Z**H,  T = Q*P*Z**H,
```

where Q and Z are unitary matrices and S and P are upper triangular.

Optionally, the unitary matrix Q from the generalized Schur factorization may be postmultiplied into an input matrix Q1, and the unitary matrix Z may be postmultiplied into an input matrix Z1. If Q1 and Z1 are the unitary matrices from ZGGHRD that reduced the matrix pair (A,B) to generalized Hessenberg form, then the output matrices Q1*Q and Z1*Z are the unitary factors from the generalized Schur factorization of (A,B):

```
A = (Q1*Q)*S*(Z1*Z)**H,  B = (Q1*Q)*P*(Z1*Z)**H.
```

To avoid overflow, eigenvalues of the matrix pair (H,T) (equivalently, of (A,B)) are computed as a pair of complex values (alpha,beta). If beta is nonzero, lambda = alpha / beta is an eigenvalue of the generalized nonsymmetric eigenvalue problem (GNEP)

```
A*x = lambda*B*x
```

and if alpha is nonzero, mu = beta / alpha is an eigenvalue of the alternate form of the GNEP

```
mu*A*y = B*y.
```

The values of alpha and beta for the i-th eigenvalue can be read directly from the generalized Schur form: alpha = S(i,i), beta = P(i,i).

Ref: C.B. Moler & G.W. Stewart, "An Algorithm for Generalized Matrix Eigenvalue Problems", SIAM J. Numer. Anal., 10(1973), pp. 241–256.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhgeqz(JOB, COMPQ, COMPZ, N, ILO, IHI, H, LDH, T, LDT, ALPHA, BETA,
                  Q, LDQ, Z, LDZ, WORK, LWORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhgeqz_(const char *job, const char *compq, const char *compz,
             const armpl_int_t *n, const armpl_int_t *ilo,
             const armpl_int_t *ihi, armpl_doublecomplex_t *h,
             const armpl_int_t *ldh, armpl_doublecomplex_t *t,
             const armpl_int_t *ldt, armpl_doublecomplex_t *alpha,
             armpl_doublecomplex_t *beta, armpl_doublecomplex_t *q,
             const armpl_int_t *ldq, armpl_doublecomplex_t *z,
             const armpl_int_t *ldz, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, double *rwork, armpl_int_t *info,
             ... );
```

Parameters

JOB Input parameter.

JOB is CHARACTER*1

= 'E': Compute eigenvalues only; = 'S': Computer eigenvalues and the Schur form.

COMPQ Input parameter.

COMPQ is CHARACTER*1

= 'N': Left Schur vectors (Q) are not computed; = 'I': Q is initialized to the unit matrix and the matrix Q of left Schur vectors of (H, T) is returned; = 'V': Q must contain a unitary matrix Q1 on entry and the product Q1* Q is returned.

COMPZ Input parameter.

COMPZ is CHARACTER*1

= 'N': Right Schur vectors (Z) are not computed; = 'I': Q is initialized to the unit matrix and the matrix Z of right Schur vectors of (H, T) is returned; = 'V': Z must contain a unitary matrix Z1 on entry and the product Z1* Z is returned.

N Input parameter.

N is INTEGER

The order of the matrices H, T, Q, and Z. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

ILO and IHI mark the rows and columns of H which are in Hessenberg form. It is assumed that A is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N. If N > 0, 1 <= ILO <= IHI <= N; if N = 0, ILO=1 and IHI=0.

H Input and output parameter.

H is COMPLEX*16

H is an array, dimension (LDH, N). On entry, the N-by-N upper Hessenberg matrix H. On exit, if JOB = 'S', H contains the upper triangular matrix S from the generalized Schur factorization. If JOB = 'E', the diagonal of H matches that of S, but the rest of H is unspecified.

LDH Input parameter.

LDH is INTEGER

The leading dimension of the array H. LDH >= max(1, N).

T Input and output parameter.

T is COMPLEX*16

T is an array, dimension (LDT, N). On entry, the N-by-N upper triangular matrix T. On exit, if JOB = 'S', T contains the upper triangular matrix P from the generalized Schur factorization. If JOB = 'E', the diagonal of T matches that of P, but the rest of T is unspecified.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

ALPHA Output parameter.

ALPHA is COMPLEX*16

ALPHA is an array, dimension (N). The complex scalars alpha that define the eigenvalues of GNEP. ALPHA(i) = S(i,i) in the generalized Schur factorization.

BETA Output parameter.

BETA is COMPLEX*16

BETA is an array, dimension (N). The real non-negative scalars beta that define the eigenvalues of GNEP. BETA(i) = P(i,i) in the generalized Schur factorization.

Together, the quantities alpha = ALPHA(j) and beta = BETA(j) represent the j-th eigenvalue of the matrix pair (A,B), in one of the forms lambda = alpha/beta or mu = beta/alpha. Since either lambda or mu may overflow, they should not, in general, be computed.

Q Input and output parameter.

Q is COMPLEX*16

Q is an array, dimension (LDQ, N). On entry, if COMPQ = 'V', the unitary matrix Q1 used in the reduction of (A,B) to generalized Hessenberg form. On exit, if COMPQ = 'I', the unitary matrix of left Schur vectors of (H, T), and if COMPQ = 'V', the unitary matrix of left Schur vectors of (A,B). Not referenced if COMPQ = 'N'.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= 1. If COMPQ='V' or 'I', then LDQ >= N.

Z Input and output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, N). On entry, if COMPZ = 'V', the unitary matrix Z1 used in the reduction of (A,B) to generalized Hessenberg form. On exit, if COMPZ = 'I', the unitary matrix of right Schur vectors of (H, T), and if COMPZ = 'V', the unitary matrix of right Schur vectors of (A,B). Not referenced if COMPZ = 'N'.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1. If COMPZ='V' or 'I', then LDZ >= N.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO >= 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N).

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value = 1,...,N: the QZ iteration did not converge. (H, T) is not in Schur form, but ALPHA(i) and BETA(i), i=INFO+1,...,N should be correct. = N+1,...,2*N: the shift calculation failed. (H, T) is not in Schur form, but ALPHA(i) and BETA(i), i=INFO-N+1,...,N should be correct.

Related Information

For this routine in other precisions, please see *chgeqz*, *dhgeqz* and *shgeqz*. It also exists with a native C interface as *LAPACKE_zhgeqz*.

4.14.60 ztgevc

ztgevc computes some or all of the right and/or left eigenvectors of a pair of complex matrices (S,P), where S and P are upper triangular. Matrix pairs of this type are produced by the generalized Schur factorization of a complex matrix pair (A,B):

```
A = Q*S*Z**H,  B = Q*P*Z**H
```

as computed by ZGGHRD + ZHGEQZ.

The right eigenvector x and the left eigenvector y of (S,P) corresponding to an eigenvalue w are defined by:

```
S*x = w*P*x,  (y**H)*S = w*(y**H)*P,
```

where y^H denotes the conjugate tranpose of y. The eigenvalues are not input to this routine, but are computed directly from the diagonal elements of S and P.

This routine returns the matrices X and/or Y of right and left eigenvectors of (S,P), or the products Z*X and/or Q*Y, where Z and Q are input matrices. If Q and Z are the unitary factors from the generalized Schur factorization of a matrix pair (A,B), then Z*X and Q*Y are the matrices of right and left eigenvectors of (A,B).

Syntax

Fortran specification:

```
use armpl_library

subroutine ztgevc(SIDE, HOWMNY, SELECT, N, S, LDS, P, LDP, VL, LDVL, VR, LDVR,
                  MM, M, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ztgevc_(const char *side, const char *howmny, const armpl_int_t *select,
             const armpl_int_t *n, const armpl_doublecomplex_t *s,
             const armpl_int_t *lds, const armpl_doublecomplex_t *p,
             const armpl_int_t *ldp, armpl_doublecomplex_t *vl,
             const armpl_int_t *ldvl, armpl_doublecomplex_t *vr,
             const armpl_int_t *ldvr, const armpl_int_t *mm, armpl_int_t *m,
             armpl_doublecomplex_t *work, double *rwork, armpl_int_t *info,
             ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'R': compute right eigenvectors only; = 'L': compute left eigenvectors only; = 'B': compute both right and left eigenvectors.

HOWMNY Input parameter.

HOWMNY is CHARACTER*1

= 'A': compute all right and/or left eigenvectors; = 'B': compute all right and/or left eigenvectors, backtransformed by the matrices in VR and/or VL; = 'S': compute selected right and/or left eigenvectors, specified by the logical array SELECT.

SELECT Input parameter.

SELECT is LOGICAL

SELECT is an array, dimension (N). If HOWMNY='S', SELECT specifies the eigenvectors to be computed. The eigenvector corresponding to the j-th eigenvalue is computed if SELECT(j) = .TRUE.. Not referenced if HOWMNY = 'A' or 'B'.

N Input parameter.

N is INTEGER

The order of the matrices S and P. N >= 0.

S Input parameter.

S is COMPLEX*16

S is an array, dimension (LDS, N). The upper triangular matrix S from a generalized Schur factorization, as computed by ZHGEQZ.

LDS Input parameter.

LDS is INTEGER

The leading dimension of array S. LDS >= max(1, N).

P Input parameter.

P is COMPLEX*16

P is an array, dimension (LDP, N). The upper triangular matrix P from a generalized Schur factorization, as computed by ZHGEQZ. P must have real diagonal elements.

LDP Input parameter.

LDP is INTEGER

The leading dimension of array P. LDP >= max(1, N).

VL Input and output parameter.

VL is COMPLEX*16

VL is an array, dimension (LDVL, MM). On entry, if SIDE = 'L' or 'B' and HOWMNY = 'B', VL must contain an N-by-N matrix Q (usually the unitary matrix Q of left Schur vectors returned by ZHGEQZ). On exit, if SIDE = 'L' or 'B', VL contains: if HOWMNY = 'A', the matrix Y of left eigenvectors of (S, P); if HOWMNY = 'B', the matrix Q*Y; if HOWMNY = 'S', the left eigenvectors of (S, P) specified by SELECT, stored consecutively in the columns of VL, in the same order as their eigenvalues. Not referenced if SIDE = 'R'.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of array VL. LDVL >= 1, and if SIDE = 'L' or 'l' or 'B' or 'b', LDVL >= N.

VR Input and output parameter.

VR is COMPLEX*16

VR is an array, dimension (LDVR, MM). On entry, if SIDE = 'R' or 'B' and HOWMNY = 'B', VR must contain an N-by-N matrix Q (usually the unitary matrix Z of right Schur vectors returned by ZHGEQZ). On exit, if SIDE = 'R' or 'B', VR contains: if HOWMNY = 'A', the matrix X of right eigenvectors of (S, P); if HOWMNY = 'B', the matrix Z*X; if HOWMNY = 'S', the right eigenvectors of (S, P) specified by SELECT, stored consecutively in the columns of VR, in the same order as their eigenvalues. Not referenced if SIDE = 'L'.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the array VR. LDVR >= 1, and if SIDE = 'R' or 'B', LDVR >= N.

MM Input parameter.

MM is INTEGER

The number of columns in the arrays VL and/or VR. MM >= M.

M Output parameter.

M is INTEGER

The number of columns in the arrays VL and/or VR actually used to store the eigenvectors. If HOWMNY = 'A' or 'B', M is set to N. Each selected eigenvector occupies one column.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *ctgevc*, *dtgevc* and *stgevc*. It also exists with a native C interface as *LAPACKE_ztgevc*.

4.14.61 ztgexc

ztgexc reorders the generalized Schur decomposition of a complex matrix pair (A,B), using an unitary equivalence transformation (A, B) := Q * (A, B) * Z^H , so that the diagonal block of (A, B) with row index IFST is moved to row ILST.

(A, B) must be in generalized Schur canonical form, that is, A and B are both upper triangular.

Optionally, the matrices Q and Z of generalized Schur vectors are updated.

```
 Q(in) * A(in) * Z(in)**H = Q(out) * A(out) * Z(out)**H
Q(in) * B(in) * Z(in)**H = Q(out) * B(out) * Z(out)**H
```

Syntax

Fortran specification:

```
use armpl_library

subroutine ztgexc(WANTQ, WANTZ, N, A, LDA, B, LDB, Q, LDQ, Z, LDZ, IFST, ILST,
                  INFO)
```

C specification:

```
#include "armpl.h"

void ztgexc_(const armpl_int_t *wantq, const armpl_int_t *wantz,
             const armpl_int_t *n, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, armpl_doublecomplex_t *q,
             const armpl_int_t *ldq, armpl_doublecomplex_t *z,
             const armpl_int_t *ldz, const armpl_int_t *ifst,
             armpl_int_t *ilst, armpl_int_t *info);
```

Parameters

WANTQ Input parameter.

WANTQ is LOGICAL

.TRUE. : update the left transformation matrix Q; .FALSE.: do not update Q.

WANTZ Input parameter.

WANTZ is LOGICAL

.TRUE. : update the right transformation matrix Z; .FALSE.: do not update Z.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the upper triangular matrix A in the pair (A, B). On exit, the updated matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). On entry, the upper triangular matrix B in the pair (A, B). On exit, the updated matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Q Input and output parameter.

Q is COMPLEX*16

Q is an array, dimension (LDQ, N). On entry, if WANTQ = .TRUE., the unitary matrix Q. On exit, the updated matrix Q. If WANTQ = .FALSE., Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= 1; If WANTQ = .TRUE., LDQ >= N.

Z Input and output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, N). On entry, if WANTZ = .TRUE., the unitary matrix Z. On exit, the updated matrix Z. If WANTZ = .FALSE., Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1; If WANTZ = .TRUE., LDZ >= N.

IFST Input parameter.

IFST is INTEGER

ILST Input and output parameter.

ILST is INTEGER

Specify the reordering of the diagonal blocks of (A, B). The block with row index IFST is moved to row ILST, by a sequence of swapping between adjacent blocks.

INFO Output parameter.

INFO is INTEGER

=0: Successful exit. <0: if INFO = -i, the i-th argument had an illegal value. =1: The transformed matrix pair (A, B) would be too far from generalized Schur form; the problem is ill- conditioned. (A, B) may have been partially reordered, and ILST points to the first row of the current position of the block being moved.

Related Information

For this routine in other precisions, please see *ctgexc*, *dtgexc* and *stgexc*. It also exists with a native C interface as *LAPACKE_ztgexc*.

4.14.62 ztgsen

ztgsen reorders the generalized Schur decomposition of a complex matrix pair (A, B) (in terms of an unitary equivalence trans- formation Q^H * (A, B) * Z), so that a selected cluster of eigenvalues appears in the leading diagonal blocks of the pair (A,B). The leading columns of Q and Z form unitary bases of the corresponding left and right eigenspaces (deflating subspaces). (A, B) must be in generalized Schur canonical form, that is, A and B are both upper triangular.

ztgsen also computes the generalized eigenvalues

```
w(j)= ALPHA(j) / BETA(j)
```

of the reordered matrix pair (A, B).

Optionally, the routine computes estimates of reciprocal condition numbers for eigenvalues and eigenspaces. These are Difu[(A11,B11), (A22,B22)] and Difl[(A11,B11), (A22,B22)], i.e. the separation(s) between the matrix pairs (A11, B11) and (A22,B22) that correspond to the selected cluster and the eigenvalues outside the cluster, resp., and norms of "projections" onto left and right eigenspaces w.r.t. the selected cluster in the (1,1)-block.

Syntax

Fortran specification:

```
use armpl_library

subroutine ztgsen(IJOB, WANTQ, WANTZ, SELECT, N, A, LDA, B, LDB, ALPHA, BETA,
                  Q, LDQ, Z, LDZ, M, PL, PR, DIF, WORK, LWORK, IWORK, LIWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void ztgsen_(const armpl_int_t *ijob, const armpl_int_t *wantq,
             const armpl_int_t *wantz, const armpl_int_t *select,
             const armpl_int_t *n, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, armpl_doublecomplex_t *alpha,
             armpl_doublecomplex_t *beta, armpl_doublecomplex_t *q,
             const armpl_int_t *ldq, armpl_doublecomplex_t *z,
             const armpl_int_t *ldz, armpl_int_t *m, double *pl, double *pr,
             double *dif, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *iwork,
             const armpl_int_t *liwork, armpl_int_t *info);
```

Parameters

IJOB Input parameter.

IJOB is INTEGER

Specifies whether condition numbers are required for the cluster of eigenvalues (PL and PR) or the deflating subspaces (Difu and Difl): =0: Only reorder w.r.t. SELECT. No extras. =1: Reciprocal of norms of "projections" onto left and right eigenspaces w.r.t. the selected cluster (PL and PR). =2: Upper bounds on Difu and Difl. F-norm-based estimate (DIF(1:2)). =3: Estimate of Difu and Difl. 1-norm-based estimate (DIF(1:2)). About 5 times as expensive as IJOB = 2. =4: Compute PL, PR and DIF (i.e. 0, 1 and 2 above): Economic version to get it all. =5: Compute PL, PR and DIF (i.e. 0, 1 and 3 above)

WANTQ Input parameter.

WANTQ is LOGICAL

.TRUE. : update the left transformation matrix Q; .FALSE.: do not update Q.

WANTZ Input parameter.

WANTZ is LOGICAL

.TRUE. : update the right transformation matrix Z; .FALSE.: do not update Z.

SELECT Input parameter.

SELECT is LOGICAL

SELECT is an array, dimension (N). SELECT specifies the eigenvalues in the selected cluster. To select an eigenvalue w(j), SELECT(j) must be set to .TRUE..

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension(LDA, N). On entry, the upper triangular matrix A, in generalized Schur canonical form. On exit, A is overwritten by the reordered matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension(LDB, N). On entry, the upper triangular matrix B, in generalized Schur canonical form. On exit, B is overwritten by the reordered matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

ALPHA Output parameter.

ALPHA is COMPLEX*16

ALPHA is an array, dimension (N) .

BETA Output parameter.

BETA is COMPLEX*16

BETA is an array, dimension (N). The diagonal elements of A and B, respectively, when the pair (A, B) has been reduced to generalized Schur form. ALPHA(i)/BETA(i) i=1,...,N are the generalized eigenvalues.

Q Input and output parameter.

Q is COMPLEX*16

Q is an array, dimension (LDQ, N). On entry, if WANTQ = .TRUE., Q is an N-by-N matrix. On exit, Q has been postmultiplied by the left unitary transformation matrix which reorder (A, B); The leading M columns of Q form orthonormal bases for the specified pair of left eigenspaces (deflating subspaces). If WANTQ = .FALSE., Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= 1. If WANTQ = .TRUE., LDQ >= N.

Z Input and output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, N). On entry, if WANTZ = .TRUE., Z is an N-by-N matrix. On exit, Z has been postmultiplied by the left unitary transformation matrix which reorder (A, B); The leading M columns of Z form orthonormal bases for the specified pair of left eigenspaces (deflating subspaces). If WANTZ = .FALSE., Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1. If WANTZ = .TRUE., LDZ >= N.

M Output parameter.

M is INTEGER

The dimension of the specified pair of left and right eigenspaces, (deflating subspaces) 0 <= M <= N.

PL Output parameter.

PL is DOUBLE PRECISION

PR Output parameter.

PR is DOUBLE PRECISION

If IJOB = 1, 4 or 5, PL, PR are lower bounds on the reciprocal of the norm of "projections" onto left and right eigenspace with respect to the selected cluster. 0 < PL, PR <= 1. If M = 0 or M = N, PL = PR = 1. If IJOB = 0, 2 or 3 PL, PR are not referenced.

DIF Output parameter.

DIF is DOUBLE PRECISION

DIF is an array, dimension (2).. If IJOB >= 2, DIF(1:2) store the estimates of Difu and Difl. If IJOB = 2 or 4, DIF(1:2) are F-norm-based upper bounds on Difu and Difl. If IJOB = 3 or 5, DIF(1:2) are 1-norm-based estimates of Difu and Difl, computed using reversed communication with ZLACN2. If M = 0 or N, DIF(1:2) = F-norm([A, B]). If IJOB = 0 or 1, DIF is not referenced.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= 1 If IJOB = 1, 2 or 4, LWORK >= 2*M*(N-M) If IJOB = 3 or 5, LWORK >= 4*M*(N-M)

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. LIWORK >= 1. If IJOB = 1, 2 or 4, LIWORK >= N+2; If IJOB = 3 or 5, LIWORK >= MAX(N+2, 2*M*(N-M));

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the IWORK array, returns this value as the first entry of the IWORK array, and no error message related to LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

=0: Successful exit. <0: If INFO = -i, the i-th argument had an illegal value. =1: Reordering of (A, B) failed because the transformed matrix pair (A, B) would be too far from generalized Schur form; the problem is very ill-conditioned. (A, B) may have been partially reordered. If requested, 0 is returned in DIF(*), PL and PR.

Related Information

For this routine in other precisions, please see *ctgsen*, *dtgsen* and *stgsen*. It also exists with a native C interface as *LAPACKE_ztgsen*.

4.14.63 ztgsna

ztgsna estimates reciprocal condition numbers for specified eigenvalues and/or eigenvectors of a matrix pair (A, B).

(A, B) must be in generalized Schur canonical form, that is, A and B are both upper triangular.

Syntax

Fortran specification:

```
use armpl_library

subroutine ztgsna(JOB, HOWMNY, SELECT, N, A, LDA, B, LDB, VL, LDVL, VR, LDVR,
                  S, DIF, MM, M, WORK, LWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ztgsna_(const char *job, const char *howmny, const armpl_int_t *select,
             const armpl_int_t *n, const armpl_doublecomplex_t *a,
             const armpl_int_t *lda, const armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, const armpl_doublecomplex_t *vl,
             const armpl_int_t *ldvl, const armpl_doublecomplex_t *vr,
             const armpl_int_t *ldvr, double *s, double *dif,
             const armpl_int_t *mm, armpl_int_t *m,
             armpl_doublecomplex_t *work, const armpl_int_t *lwork,
             armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

JOB Input parameter.

JOB is CHARACTER*1

Specifies whether condition numbers are required for eigenvalues (S) or eigenvectors (DIF): = 'E': for eigenvalues only (S); = 'V': for eigenvectors only (DIF); = 'B': for both eigenvalues and eigenvectors (S and DIF).

HOWMNY Input parameter.

HOWMNY is CHARACTER*1

= 'A': compute condition numbers for all eigenpairs; = 'S': compute condition numbers for selected eigenpairs specified by the array SELECT.

SELECT Input parameter.

SELECT is LOGICAL

SELECT is an array, dimension (N). If HOWMNY = 'S', SELECT specifies the eigenpairs for which condition numbers are required. To select condition numbers for the corresponding j-th eigenvalue and/or eigenvector, SELECT(j) must be set to .TRUE.. If HOWMNY = 'A', SELECT is not referenced.

N Input parameter.

N is INTEGER

The order of the square matrix pair (A, B). N >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The upper triangular matrix A in the pair (A, B).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). The upper triangular matrix B in the pair (A, B).

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

VL Input parameter.

VL is COMPLEX*16

VL is an array, dimension (LDVL, M). IF JOB = 'E' or 'B', VL must contain left eigenvectors of (A, B), corresponding to the eigenpairs specified by HOWMNY and SELECT. The eigenvectors must be stored in consecutive columns of VL, as returned by ZTGEVC. If JOB = 'V', VL is not referenced.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the array VL. LDVL >= 1; and If JOB = 'E' or 'B', LDVL >= N.

VR Input parameter.

VR is COMPLEX*16

VR is an array, dimension (LDVR, M). IF JOB = 'E' or 'B', VR must contain right eigenvectors of (A, B), corresponding to the eigenpairs specified by HOWMNY and SELECT. The eigenvectors must be stored in consecutive columns of VR, as returned by ZTGEVC. If JOB = 'V', VR is not referenced.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the array VR. LDVR >= 1; If JOB = 'E' or 'B', LDVR >= N.

S Output parameter.

S is DOUBLE PRECISION

S is an array, dimension (MM). If JOB = 'E' or 'B', the reciprocal condition numbers of the selected eigenvalues, stored in consecutive elements of the array. If JOB = 'V', S is not referenced.

DIF Output parameter.

DIF is DOUBLE PRECISION

DIF is an array, dimension (MM). If JOB = 'V' or 'B', the estimated reciprocal condition numbers of the selected eigenvectors, stored in consecutive elements of the array. If the eigenvalues cannot be reordered to compute DIF(j), DIF(j) is set to 0; this can only occur when the true value would be very small anyway. For each eigenvalue/vector specified by SELECT, DIF stores a Frobenius norm-based estimate of Difl. If JOB = 'E', DIF is not referenced.

MM Input parameter.

MM is INTEGER

The number of elements in the arrays S and DIF. MM >= M.

M Output parameter.

M is INTEGER

The number of elements of the arrays S and DIF used to store the specified condition numbers; for each selected eigenvalue one element is used. If HOWMNY = 'A', M is set to N.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= max(1, N). If JOB = 'V' or 'B', LWORK >= max(1,2*N*N).

IWORK Output parameter.

IWORK is INTEGER array, dimension (N+2)

If JOB = 'E', IWORK is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit < 0: If INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctgsna*, *dtgsna* and *stgsna*. It also exists with a native C interface as *LAPACKE_ztgsna*.

4.14.64 ztgsyl

`ztgsyl` solves the generalized Sylvester equation:

```
 A * R - L * B = scale * C            (1)
D * R - L * E = scale * F
```

where R and L are unknown m-by-n matrices, (A, D), (B, E) and (C, F) are given matrix pairs of size m-by-m, n-by-n and m-by-n, respectively, with complex entries. A, B, D and E are upper triangular (i.e., (A,D) and (B,E) in generalized Schur form).

The solution (R, L) overwrites (C, F). 0 <= SCALE <= 1 is an output scaling factor chosen to avoid overflow.

In matrix notation (1) is equivalent to solve Zx = scale*b, where Z is defined as

```
Z = [ kron(In, A)  -kron(B**H, Im) ]         (2)
    [ kron(In, D)  -kron(E**H, Im) ],
```

Here Ix is the identity matrix of size x and X^H is the conjugate transpose of X. Kron(X, Y) is the Kronecker product between the matrices X and Y.

If TRANS = 'C', y in the conjugate transposed system Z^H *y = scale*b is solved for, which is equivalent to solve for R and L in

```
 A**H * R + D**H * L = scale * C            (3)
R * B**H + L * E**H = scale * -F
```

This case (TRANS = 'C') is used to compute an one-norm-based estimate of Dif[(A,D), (B,E)], the separation between the matrix pairs (A,D) and (B,E), using ZLACON.

If IJOB >= 1, `ztgsyl` computes a Frobenius norm-based estimate of Dif[(A,D),(B,E)]. That is, the reciprocal of a lower bound on the reciprocal of the smallest singular value of Z.

This is a level-3 BLAS algorithm.

Syntax

Fortran specification:

```
use armpl_library

subroutine ztgsyl(TRANS, IJOB, M, N, A, LDA, B, LDB, C, LDC, D, LDD, E, LDE,
                  F, LDF, SCALE, DIF, WORK, LWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ztgsyl_(const char *trans, const armpl_int_t *ijob, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_doublecomplex_t *a,
             const armpl_int_t *lda, const armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, armpl_doublecomplex_t *c,
             const armpl_int_t *ldc, const armpl_doublecomplex_t *d,
             const armpl_int_t *ldd, const armpl_doublecomplex_t *e,
             const armpl_int_t *lde, armpl_doublecomplex_t *f,
             const armpl_int_t *ldf, double *scale, double *dif,
             armpl_doublecomplex_t *work, const armpl_int_t *lwork,
             armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': solve the generalized sylvester equation (1). = 'C': solve the "conjugate transposed" system (3).

IJOB Input parameter.

IJOB is INTEGER

Specifies what kind of functionality to be performed. =0: solve (1) only. =1: The functionality of 0 and 3. =2: The functionality of 0 and 4. =3: Only an estimate of Dif[(A, D), (B, E)] is computed. (look ahead strategy is used). =4: Only an estimate of Dif[(A, D), (B, E)] is computed. (ZGECON on sub-systems is used). Not referenced if TRANS = 'C'.

M Input parameter.

M is INTEGER

The order of the matrices A and D, and the row dimension of the matrices C, F, R and L.

N Input parameter.

N is INTEGER

The order of the matrices B and E, and the column dimension of the matrices C, F, R and L.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, M). The upper triangular matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). The upper triangular matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). On entry, C contains the right-hand-side of the first matrix equation in (1) or (3). On exit, if IJOB = 0, 1 or 2, C has been overwritten by the solution R. If IJOB = 3 or 4 and TRANS = 'N', C holds R, the solution achieved during the computation of the Dif-estimate.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

D Input parameter.

D is COMPLEX*16

D is an array, dimension (LDD, M). The upper triangular matrix D.

LDD Input parameter.

LDD is INTEGER

The leading dimension of the array D. LDD >= max(1, M).

E Input parameter.

E is COMPLEX*16

E is an array, dimension (LDE, N). The upper triangular matrix E.

LDE Input parameter.

LDE is INTEGER

The leading dimension of the array E. LDE >= max(1, N).

F Input and output parameter.

F is COMPLEX*16

F is an array, dimension (LDF, N). On entry, F contains the right-hand-side of the second matrix equation in (1) or (3). On exit, if IJOB = 0, 1 or 2, F has been overwritten by the solution L. If IJOB = 3 or 4 and TRANS = 'N', F holds L, the solution achieved during the computation of the Dif-estimate.

LDF Input parameter.

LDF is INTEGER

The leading dimension of the array F. LDF >= max(1, M).

DIF Output parameter.

DIF is DOUBLE PRECISION

On exit DIF is the reciprocal of a lower bound of the reciprocal of the Dif-function, i.e. DIF is an upper bound of Dif[(A, D), (B, E)] = sigma-min(Z), where Z as in (2). IF IJOB = 0 or TRANS = 'C', DIF is not referenced.

SCALE Output parameter.

SCALE is DOUBLE PRECISION

On exit SCALE is the scaling factor in (1) or (3). If 0 < SCALE < 1, C and F hold the solutions R and L, resp., to a slightly perturbed system but the input matrices A, B, D and E have not been changed. If SCALE = 0, R and L will hold the solutions to the homogenious system with C = F = 0.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK > = 1. If IJOB = 1 or 2 and TRANS = 'N', LWORK >= max(1,2*M*N).

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (M+N+2)

INFO Output parameter.

INFO is INTEGER

=0: successful exit <0: If INFO = -i, the i-th argument had an illegal value. >0: (A, D) and (B, E) have common or very close eigenvalues.

Related Information

For this routine in other precisions, please see *ctgsyl*, *dtgsyl* and *stgsyl*. It also exists with a native C interface as *LAPACKE_ztgsyl*.

4.15 LAPACK condition number estimation routines

4.15.1 cgbcon

`cgbcon` estimates the reciprocal of the condition number of a complex general band matrix A, in either the 1-norm or the infinity-norm, using the LU factorization computed by CGBTRF.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as

```
RCOND = 1 / ( norm(A) * norm(inv(A)) ).
```

Syntax

Fortran specification:

```
use armpl_library

subroutine cgbcon(NORM, N, KL, KU, AB, LDAB, IPIV, ANORM, RCOND, WORK, RWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void cgbcon_(const char *norm, const armpl_int_t *n, const armpl_int_t *kl,
             const armpl_int_t *ku, const armpl_singlecomplex_t *ab,
             const armpl_int_t *ldab, const armpl_int_t *ipiv,
             const float *anorm, float *rcond, armpl_singlecomplex_t *work,
             float *rwork, armpl_int_t *info, ... );
```

Parameters

NORM Input parameter.

`NORM` is `CHARACTER*1`

Specifies whether the 1-norm condition number or the infinity-norm condition number is required: = ‘1’ or ‘O’: 1-norm; = ‘I’: Infinity-norm.

N Input parameter.

`N` is `INTEGER`

The order of the matrix A. `N` >= 0.

KL Input parameter.

`KL` is `INTEGER`

The number of subdiagonals within the band of A. `KL` >= 0.

KU Input parameter.

`KU` is `INTEGER`

The number of superdiagonals within the band of A. `KU` >= 0.

AB Input parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). Details of the LU factorization of the band matrix A, as computed by CGBTRF. U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= 2*KL+KU+1.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices; for 1 <= i <= N, row i of the matrix was interchanged with row IPIV(i).

ANORM Input parameter.

ANORM is REAL

If NORM = '1' or 'O', the 1-norm of the original matrix A. If NORM = 'I', the infinity-norm of the original matrix A.

RCOND Output parameter.

RCOND is REAL

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(norm(A) * norm(inv(A))).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dgbcon*, *sgbcon* and *zgbcon*. It also exists with a native C interface as *LAPACKE_cgbcon*.

4.15.2 cgecon

cgecon estimates the reciprocal of the condition number of a general complex matrix A, in either the 1-norm or the infinity-norm, using the LU factorization computed by CGETRF.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as

```
RCOND = 1 / ( norm(A) * norm(inv(A)) ).
```

Syntax

Fortran specification:

```
use armpl_library

subroutine cgecon(NORM, N, A, LDA, ANORM, RCOND, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgecon_(const char *norm, const armpl_int_t *n,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const float *anorm, float *rcond, armpl_singlecomplex_t *work,
             float *rwork, armpl_int_t *info, ... );
```

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies whether the 1-norm condition number or the infinity-norm condition number is required: = '1' or 'O': 1-norm; = 'I': Infinity-norm.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The factors L and U from the factorization A = P*L*U as computed by CGETRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

ANORM Input parameter.

ANORM is REAL

If NORM = '1' or 'O', the 1-norm of the original matrix A. If NORM = 'I', the infinity-norm of the original matrix A.

RCOND Output parameter.

RCOND is REAL

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(norm(A) * norm(inv(A))).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dgecon*, *sgecon* and *zgecon*. It also exists with a native C interface as *LAPACKE_cgecon*.

4.15.3 cgtcon

cgtcon estimates the reciprocal of the condition number of a complex tridiagonal matrix A using the LU factorization as computed by CGTTRF.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))).

Syntax

Fortran specification:

```
use armpl_library

subroutine cgtcon(NORM, N, DL, D, DU, DU2, IPIV, ANORM, RCOND, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgtcon_(const char *norm, const armpl_int_t *n,
             const armpl_singlecomplex_t *dl, const armpl_singlecomplex_t *d,
             const armpl_singlecomplex_t *du,
             const armpl_singlecomplex_t *du2, const armpl_int_t *ipiv,
             const float *anorm, float *rcond, armpl_singlecomplex_t *work,
             armpl_int_t *info, ... );
```

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies whether the 1-norm condition number or the infinity-norm condition number is required: = '1' or 'O': 1-norm; = 'I': Infinity-norm.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

DL Input parameter.

DL is COMPLEX

DL is an array, dimension (N-1). The (n-1) multipliers that define the matrix L from the LU factorization of A as computed by CGTTRF.

D Input parameter.

D is COMPLEX

D is an array, dimension (N). The n diagonal elements of the upper triangular matrix U from the LU factorization of A.

DU Input parameter.

DU is COMPLEX

DU is an array, dimension (N-1). The (n-1) elements of the first superdiagonal of U.

DU2 Input parameter.

DU2 is COMPLEX

DU2 is an array, dimension (N-2). The (n-2) elements of the second superdiagonal of U.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices; for 1 <= i <= n, row i of the matrix was interchanged with row IPIV(i). IPIV(i) will always be either i or i+1; IPIV(i) = i indicates a row interchange was not required.

ANORM Input parameter.

ANORM is REAL

If NORM = '1' or 'O', the 1-norm of the original matrix A. If NORM = 'I', the infinity-norm of the original matrix A.

RCOND Output parameter.

RCOND is REAL

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dgtcon*, *sgtcon* and *zgtcon*. It also exists with a native C interface as *LAPACKE_cgtcon*.

4.15.4 checon

checon estimates the reciprocal of the condition number of a complex Hermitian matrix A using the factorization A = U*D*U^H or A = L*D*L^H computed by CHETRF.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))).

Syntax

Fortran specification:

```
use armpl_library

subroutine checon(UPLO, N, A, LDA, IPIV, ANORM, RCOND, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void checon_(const char *uplo, const armpl_int_t *n,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const armpl_int_t *ipiv, const float *anorm, float *rcond,
             armpl_singlecomplex_t *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^H ; = 'L': Lower triangular, form is A = L*D*L^H .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CHETRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CHETRF.

ANORM Input parameter.

ANORM is REAL

The 1-norm of the original matrix A.

RCOND Output parameter.

RCOND is REAL

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zhecon*. It also exists with a native C interface as *LAPACKE_checon*.

4.15.5 checon_3

CHECON_3 estimates the reciprocal of the condition number (in the 1-norm) of a complex Hermitian matrix A using the factorization computed by CHETRF_RK or CHETRF_BK:

```
A = P*U*D*(U**H)*(P**T) or A = P*L*D*(L**H)*(P**T),
```

where U (or L) is unit upper (or lower) triangular matrix, U^H (or L^H) is the conjugate of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is Hermitian and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))). This routine uses BLAS3 solver CHETRS_3.

Syntax

Fortran specification:

```
use armpl_library

subroutine checon_3(UPLO, N, A, LDA, E, IPIV, ANORM, RCOND, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void checon_3_(const char *uplo, const armpl_int_t *n,
               const armpl_singlecomplex_t *a, const armpl_int_t *lda,
               const armpl_singlecomplex_t *e, const armpl_int_t *ipiv,
               const float *anorm, float *rcond, armpl_singlecomplex_t *work,
               armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix: = 'U': Upper triangular, form is A = P*U*D*(U^H)*(P^T); = 'L': Lower triangular, form is A = P*L*D*(L^H)*(P^T).

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). Diagonal of the block diagonal matrix D and factors U or L as computed by CHETRF_RK and CHETRF_BK: a) ONLY diagonal elements of the Hermitian block diagonal matrix D

on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D should be provided on entry in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Input parameter.

E is COMPLEX

E is an array, dimension (N). On entry, contains the superdiagonal (or subdiagonal) elements of the Hermitian block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i),i=2:N, E(1) not referenced; If UPLO = 'L': E(i) = D(i+1,i),i=1:N-1, E(N) not referenced.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is not referenced in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CHETRF_RK or CHETRF_BK.

ANORM Input parameter.

ANORM is REAL

The 1-norm of the original matrix A.

RCOND Output parameter.

RCOND is REAL

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zhecon_3*. It also exists with a native C interface as *LAPACKE_checon_3*.

4.15.6 checon_rook

CHECON_ROOK estimates the reciprocal of the condition number of a complex Hermitian matrix A using the factorization A = U*D*U^H or A = L*D*L^H computed by CHETRF_ROOK.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))).

Syntax

Fortran specification:

```
use armpl_library

subroutine checon_rook(UPLO, N, A, LDA, IPIV, ANORM, RCOND, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void checon_rook_(const char *uplo, const armpl_int_t *n,
                  const armpl_singlecomplex_t *a, const armpl_int_t *lda,
                  const armpl_int_t *ipiv, const float *anorm, float *rcond,
                  armpl_singlecomplex_t *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^H ; = 'L': Lower triangular, form is A = L*D*L^H .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CHETRF_ROOK.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CHETRF_ROOK.

ANORM Input parameter.

ANORM is REAL

The 1-norm of the original matrix A.

RCOND Output parameter.

RCOND is REAL

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zhecon_rook*.

4.15.7 chpcon

chpcon estimates the reciprocal of the condition number of a complex Hermitian packed matrix A using the factorization A = U*D*U^H or A = L*D*L^H computed by CHPTRF.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))).

Syntax

Fortran specification:

```
use armpl_library

subroutine chpcon(UPLO, N, AP, IPIV, ANORM, RCOND, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void chpcon_(const char *uplo, const armpl_int_t *n,
             const armpl_singlecomplex_t *ap, const armpl_int_t *ipiv,
             const float *anorm, float *rcond, armpl_singlecomplex_t *work,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^H ; = 'L': Lower triangular, form is A = L*D*L^H .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CHPTRF, stored as a packed triangular matrix.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CHPTRF.

ANORM Input parameter.

ANORM is REAL

The 1-norm of the original matrix A.

RCOND Output parameter.

RCOND is REAL

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zhpcon*. It also exists with a native C interface as *LAPACKE_chpcon*.

4.15.8 cpbcon

cpbcon estimates the reciprocal of the condition number (in the 1-norm) of a complex Hermitian positive definite band matrix using the Cholesky factorization A = U^H *U or A = L*L^H computed by CPBTRF.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))).

Syntax

Fortran specification:

```
use armpl_library

subroutine cpbcon(UPLO, N, KD, AB, LDAB, ANORM, RCOND, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cpbcon_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
             const armpl_singlecomplex_t *ab, const armpl_int_t *ldab,
             const float *anorm, float *rcond, armpl_singlecomplex_t *work,
             float *rwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangular factor stored in AB; = 'L': Lower triangular factor stored in AB.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of sub-diagonals if UPLO = 'L'. KD >= 0.

AB Input parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). The triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H of the band matrix A, stored in the first KD+1 rows of the array. The j-th column of U or L is stored in the j-th column of the array AB as follows: if UPLO ='U', AB(kd+1+i-j,j) = U(i,j) for max(1,j-kd)<=i<=j; if UPLO ='L', AB(1+i-j,j) = L(i,j) for j<=i<=min(n,j+kd).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

ANORM Input parameter.

ANORM is REAL

The 1-norm (or infinity-norm) of the Hermitian band matrix A.

RCOND Output parameter.

RCOND is REAL

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dpbcon*, *spbcon* and *zpbcon*. It also exists with a native C interface as *LAPACKE_cpbcon*.

4.15.9 cpocon

cpocon estimates the reciprocal of the condition number (in the 1-norm) of a complex Hermitian positive definite matrix using the Cholesky factorization A = U^H *U or A = L*L^H computed by CPOTRF.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))).

Syntax

Fortran specification:

```
use armpl_library

subroutine cpocon(UPLO, N, A, LDA, ANORM, RCOND, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cpocon_(const char *uplo, const armpl_int_t *n,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const float *anorm, float *rcond, armpl_singlecomplex_t *work,
             float *rwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H , as computed by CPOTRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

ANORM Input parameter.

ANORM is REAL

The 1-norm (or infinity-norm) of the Hermitian matrix A.

RCOND Output parameter.

RCOND is REAL

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dpocon*, *spocon* and *zpocon*. It also exists with a native C interface as *LAPACKE_cpocon*.

4.15.10 cppcon

cppcon estimates the reciprocal of the condition number (in the 1-norm) of a complex Hermitian positive definite packed matrix using the Cholesky factorization A = U^H *U or A = L*L^H computed by CPPTRF.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))).

Syntax

Fortran specification:

```
use armpl_library

subroutine cppcon(UPLO, N, AP, ANORM, RCOND, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cppcon_(const char *uplo, const armpl_int_t *n,
             const armpl_singlecomplex_t *ap, const float *anorm,
             float *rcond, armpl_singlecomplex_t *work, float *rwork,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). The triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H , packed columnwise in a linear array. The j-th column of U or L is stored in the array AP as

follows: if UPLO = 'U', AP(i + (j-1)*j/2) = U(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = L(i,j) for j<=i<=n.

ANORM Input parameter.

ANORM is REAL

The 1-norm (or infinity-norm) of the Hermitian matrix A.

RCOND Output parameter.

RCOND is REAL

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dppcon*, *sppcon* and *zppcon*. It also exists with a native C interface as *LAPACKE_cppcon*.

4.15.11 cptcon

cptcon computes the reciprocal of the condition number (in the 1-norm) of a complex Hermitian positive definite tridiagonal matrix using the factorization A = L*D*L^H or A = U^H *D*U computed by CPTTRF.

Norm(inv(A)) is computed by a direct method, and the reciprocal of the condition number is computed as

```
RCOND = 1 / (ANORM * norm(inv(A))).
```

Syntax

Fortran specification:

```
use armpl_library

subroutine cptcon(N, D, E, ANORM, RCOND, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cptcon_(const armpl_int_t *n, const float *d,
             const armpl_singlecomplex_t *e, const float *anorm, float *rcond,
             float *rwork, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

D Input parameter.

D is REAL

D is an array, dimension (N). The n diagonal elements of the diagonal matrix D from the factorization of A, as computed by CPTTRF.

E Input parameter.

E is COMPLEX

E is an array, dimension (N-1). The (n-1) off-diagonal elements of the unit bidiagonal factor U or L from the factorization of A, as computed by CPTTRF.

ANORM Input parameter.

ANORM is REAL

The 1-norm of the original matrix A.

RCOND Output parameter.

RCOND is REAL

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is the 1-norm of inv(A) computed in this routine.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dptcon*, *sptcon* and *zptcon*. It also exists with a native C interface as *LAPACKE_cptcon*.

4.15.12 cspcon

cspcon estimates the reciprocal of the condition number (in the 1-norm) of a complex symmetric packed matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by CSPTRF.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))).

Syntax

Fortran specification:

```
use armpl_library

subroutine cspcon(UPLO, N, AP, IPIV, ANORM, RCOND, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void cspcon_(const char *uplo, const armpl_int_t *n,
             const armpl_singlecomplex_t *ap, const armpl_int_t *ipiv,
             const float *anorm, float *rcond, armpl_singlecomplex_t *work,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CSPTRF, stored as a packed triangular matrix.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CSPTRF.

ANORM Input parameter.

ANORM is REAL

The 1-norm of the original matrix A.

RCOND Output parameter.

RCOND is REAL

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dspcon*, *sspcon* and *zspcon*. It also exists with a native C interface as *LAPACKE_cspcon*.

4.15.13 csycon

`csycon` estimates the reciprocal of the condition number (in the 1-norm) of a complex symmetric matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by CSYTRF.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))).

Syntax

Fortran specification:

```
use armpl_library

subroutine csycon(UPLO, N, A, LDA, IPIV, ANORM, RCOND, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void csycon_(const char *uplo, const armpl_int_t *n,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const armpl_int_t *ipiv, const float *anorm, float *rcond,
             armpl_singlecomplex_t *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = '`U`': Upper triangular, form is `A` = U*D*U^T ; = '`L`': Lower triangular, form is `A` = L*D*L^T .

N Input parameter.

`N` is `INTEGER`

The order of the matrix `A`. `N` >= 0.

A Input parameter.

`A` is `COMPLEX`

`A` is an array, dimension (`LDA`, `N`). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CSYTRF.

LDA Input parameter.

`LDA` is `INTEGER`

The leading dimension of the array `A`. `LDA` >= max(1, `N`).

IPIV Input parameter.

`IPIV` is `INTEGER` array, dimension (`N`)

Details of the interchanges and the block structure of D as determined by CSYTRF.

ANORM Input parameter.

ANORM is REAL

The 1-norm of the original matrix A.

RCOND Output parameter.

RCOND is REAL

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dsycon*, *ssycon* and *zsycon*. It also exists with a native C interface as *LAPACKE_csycon*.

4.15.14 csycon_3

CSYCON_3 estimates the reciprocal of the condition number (in the 1-norm) of a complex symmetric matrix A using the factorization computed by CSYTRF_RK or CSYTRF_BK:

```
A = P*U*D*(U**T)*(P**T) or A = P*L*D*(L**T)*(P**T),
```

where U (or L) is unit upper (or lower) triangular matrix, U^T (or L^T) is the transpose of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))). This routine uses BLAS3 solver CSYTRS_3.

Syntax

Fortran specification:

```
use armpl_library

subroutine csycon_3(UPLO, N, A, LDA, E, IPIV, ANORM, RCOND, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void csycon_3_(const char *uplo, const armpl_int_t *n,
               const armpl_singlecomplex_t *a, const armpl_int_t *lda,
               const armpl_singlecomplex_t *e, const armpl_int_t *ipiv,
               const float *anorm, float *rcond, armpl_singlecomplex_t *work,
               armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix: = 'U': Upper triangular, form is A = P*U*D*(U^T)*(P^T); = 'L': Lower triangular, form is A = P*L*D*(L^T)*(P^T).

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). Diagonal of the block diagonal matrix D and factors U or L as computed by CSYTRF_RK and CSYTRF_BK: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D should be provided on entry in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Input parameter.

E is COMPLEX

E is an array, dimension (N). On entry, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i),i=2:N, E(1) not referenced; If UPLO = 'L': E(i) = D(i+1,i),i=1:N-1, E(N) not referenced.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is not referenced in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CSYTRF_RK or CSYTRF_BK.

ANORM Input parameter.

ANORM is REAL

The 1-norm of the original matrix A.

RCOND Output parameter.

RCOND is REAL

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dsycon_3*, *ssycon_3* and *zsycon_3*. It also exists with a native C interface as *LAPACKE_csycon_3*.

4.15.15 csycon_rook

CSYCON_ROOK estimates the reciprocal of the condition number (in the 1-norm) of a complex symmetric matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by CSYTRF_ROOK.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))).

Syntax

Fortran specification:

```
use armpl_library

subroutine csycon_rook(UPLO, N, A, LDA, IPIV, ANORM, RCOND, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void csycon_rook_(const char *uplo, const armpl_int_t *n,
                  const armpl_singlecomplex_t *a, const armpl_int_t *lda,
                  const armpl_int_t *ipiv, const float *anorm, float *rcond,
                  armpl_singlecomplex_t *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CSYTRF_ROOK.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CSYTRF_ROOK.

ANORM Input parameter.

ANORM is REAL

The 1-norm of the original matrix A.

RCOND Output parameter.

RCOND is REAL

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dsycon_rook*, *ssycon_rook* and *zsycon_rook*.

4.15.16 ctbcon

ctbcon estimates the reciprocal of the condition number of a triangular band matrix A, in either the 1-norm or the infinity-norm.

The norm of A is computed and an estimate is obtained for norm(inv(A)), then the reciprocal of the condition number is computed as

```
RCOND = 1 / ( norm(A) * norm(inv(A)) ).
```

Syntax

Fortran specification:

```
use armpl_library

subroutine ctbcon(NORM, UPLO, DIAG, N, KD, AB, LDAB, RCOND, WORK, RWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void ctbcon_(const char *norm, const char *uplo, const char *diag,
             const armpl_int_t *n, const armpl_int_t *kd,
             const armpl_singlecomplex_t *ab, const armpl_int_t *ldab,
             float *rcond, armpl_singlecomplex_t *work, float *rwork,
             armpl_int_t *info, ... );
```

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies whether the 1-norm condition number or the infinity-norm condition number is required: = '1' or 'O': 1-norm; = 'I': Infinity-norm.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals or subdiagonals of the triangular band matrix A. KD >= 0.

AB Input parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). The upper or lower triangular band matrix A, stored in the first kd+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd). If DIAG = 'U', the diagonal elements of A are not referenced and are assumed to be 1.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

RCOND Output parameter.

RCOND is REAL

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(norm(A) * norm(inv(A))).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dtbcon*, *stbcon* and *ztbcon*. It also exists with a native C interface as *LAPACKE_ctbcon*.

4.15.17 ctpcon

`ctpcon` estimates the reciprocal of the condition number of a packed triangular matrix A, in either the 1-norm or the infinity-norm.

The norm of A is computed and an estimate is obtained for norm(inv(A)), then the reciprocal of the condition number is computed as

```
RCOND = 1 / ( norm(A) * norm(inv(A)) ).
```

Syntax

Fortran specification:

```
use armpl_library

subroutine ctpcon(NORM, UPLO, DIAG, N, AP, RCOND, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ctpcon_(const char *norm, const char *uplo, const char *diag,
             const armpl_int_t *n, const armpl_singlecomplex_t *ap,
             float *rcond, armpl_singlecomplex_t *work, float *rwork,
             armpl_int_t *info, ... );
```

Parameters

NORM Input parameter.

`NORM` is `CHARACTER*1`

Specifies whether the 1-norm condition number or the infinity-norm condition number is required: = '1' or 'O': 1-norm; = 'I': Infinity-norm.

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

= '`U`': A is upper triangular; = '`L`': A is lower triangular.

DIAG Input parameter.

`DIAG` is `CHARACTER*1`

= '`N`': A is non-unit triangular; = '`U`': A is unit triangular.

N Input parameter.

`N` is `INTEGER`

The order of the matrix A. `N` >= 0.

AP Input parameter.

`AP` is `COMPLEX`

AP is an array, dimension (N*(N+1)/2). The upper or lower triangular matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n. If DIAG = 'U', the diagonal elements of A are not referenced and are assumed to be 1.

RCOND Output parameter.

RCOND is REAL

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(norm(A) * norm(inv(A))).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dtpcon*, *stpcon* and *ztpcon*. It also exists with a native C interface as *LAPACKE_ctpcon*.

4.15.18 ctrcon

ctrcon estimates the reciprocal of the condition number of a triangular matrix A, in either the 1-norm or the infinity-norm.

The norm of A is computed and an estimate is obtained for norm(inv(A)), then the reciprocal of the condition number is computed as

```
RCOND = 1 / ( norm(A) * norm(inv(A)) ).
```

Syntax

Fortran specification:

```
use armpl_library

subroutine ctrcon(NORM, UPLO, DIAG, N, A, LDA, RCOND, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ctrcon_(const char *norm, const char *uplo, const char *diag,
             const armpl_int_t *n, const armpl_singlecomplex_t *a,
             const armpl_int_t *lda, float *rcond,
             armpl_singlecomplex_t *work, float *rwork, armpl_int_t *info,
             ... );
```

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies whether the 1-norm condition number or the infinity-norm condition number is required: = '1' or 'O': 1-norm; = 'I': Infinity-norm.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

RCOND Output parameter.

RCOND is REAL

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(norm(A) * norm(inv(A))).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dtrcon*, *strcon* and *ztrcon*. It also exists with a native C interface as *LAPACKE_ctrcon*.

4.15.19 dgbcon

dgbcon estimates the reciprocal of the condition number of a real general band matrix A, in either the 1-norm or the infinity-norm, using the LU factorization computed by DGBTRF.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as

```
RCOND = 1 / ( norm(A) * norm(inv(A)) ).
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dgbcon(NORM, N, KL, KU, AB, LDAB, IPIV, ANORM, RCOND, WORK, IWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dgbcon_(const char *norm, const armpl_int_t *n, const armpl_int_t *kl,
             const armpl_int_t *ku, const double *ab, const armpl_int_t *ldab,
             const armpl_int_t *ipiv, const double *anorm, double *rcond,
             double *work, armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies whether the 1-norm condition number or the infinity-norm condition number is required: = '1' or 'O': 1-norm; = 'I': Infinity-norm.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

AB Input parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). Details of the LU factorization of the band matrix A, as computed by DGBTRF. U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= 2*KL+KU+1.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices; for 1 <= i <= N, row i of the matrix was interchanged with row IPIV(i).

ANORM Input parameter.

ANORM is DOUBLE PRECISION

If NORM = '1' or 'O', the 1-norm of the original matrix A. If NORM = 'I', the infinity-norm of the original matrix A.

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(norm(A) * norm(inv(A))).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgbcon*, *sgbcon* and *zgbcon*. It also exists with a native C interface as *LAPACKE_dgbcon*.

4.15.20 dgecon

dgecon estimates the reciprocal of the condition number of a general real matrix A, in either the 1-norm or the infinity-norm, using the LU factorization computed by DGETRF.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as

```
RCOND = 1 / ( norm(A) * norm(inv(A)) ).
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dgecon(NORM, N, A, LDA, ANORM, RCOND, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgecon_(const char *norm, const armpl_int_t *n, const double *a,
             const armpl_int_t *lda, const double *anorm, double *rcond,
             double *work, armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies whether the 1-norm condition number or the infinity-norm condition number is required: = '1' or 'O': 1-norm; = 'I': Infinity-norm.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). The factors L and U from the factorization A = P*L*U as computed by DGETRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

ANORM Input parameter.

ANORM is DOUBLE PRECISION

If NORM = '1' or 'O', the 1-norm of the original matrix A. If NORM = 'I', the infinity-norm of the original matrix A.

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(norm(A) * norm(inv(A))).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (4*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgecon*, *sgecon* and *zgecon*. It also exists with a native C interface as *LAPACKE_dgecon*.

4.15.21 dgtcon

`dgtcon` estimates the reciprocal of the condition number of a real tridiagonal matrix A using the LU factorization as computed by DGTTRF.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))).

Syntax

Fortran specification:

```
use armpl_library

subroutine dgtcon(NORM, N, DL, D, DU, DU2, IPIV, ANORM, RCOND, WORK, IWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dgtcon_(const char *norm, const armpl_int_t *n, const double *dl,
             const double *d, const double *du, const double *du2,
             const armpl_int_t *ipiv, const double *anorm, double *rcond,
             double *work, armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

NORM Input parameter.

`NORM` is `CHARACTER*1`

Specifies whether the 1-norm condition number or the infinity-norm condition number is required: = '1' or 'O': 1-norm; = 'I': Infinity-norm.

N Input parameter.

`N` is `INTEGER`

The order of the matrix A. `N` >= 0.

DL Input parameter.

`DL` is `DOUBLE PRECISION`

`DL` is an array, dimension (N-1). The (n-1) multipliers that define the matrix L from the LU factorization of A as computed by DGTTRF.

D Input parameter.

`D` is `DOUBLE PRECISION`

`D` is an array, dimension (`N`). The n diagonal elements of the upper triangular matrix U from the LU factorization of A.

DU Input parameter.

`DU` is `DOUBLE PRECISION`

`DU` is an array, dimension (N-1). The (n-1) elements of the first superdiagonal of U.

DU2 Input parameter.

`DU2` is `DOUBLE PRECISION`

`DU2` is an array, dimension (N-2). The (n-2) elements of the second superdiagonal of U.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices; for 1 <= i <= n, row i of the matrix was interchanged with row IPIV(i). IPIV(i) will always be either i or i+1; IPIV(i) = i indicates a row interchange was not required.

ANORM Input parameter.

ANORM is DOUBLE PRECISION

If NORM = '1' or 'O', the 1-norm of the original matrix A. If NORM = 'I', the infinity-norm of the original matrix A.

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (2*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgtcon*, *sgtcon* and *zgtcon*. It also exists with a native C interface as *LAPACKE_dgtcon*.

4.15.22 dpbcon

dpbcon estimates the reciprocal of the condition number (in the 1-norm) of a real symmetric positive definite band matrix using the Cholesky factorization A = U^T *U or A = L*L^T computed by DPBTRF.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))).

Syntax

Fortran specification:

```
use armpl_library

subroutine dpbcon(UPLO, N, KD, AB, LDAB, ANORM, RCOND, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dpbcon_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
             const double *ab, const armpl_int_t *ldab, const double *anorm,
```

(continues on next page)

(continued from previous page)

```
                double *rcond, double *work, armpl_int_t *iwork,
                armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangular factor stored in AB; = 'L': Lower triangular factor stored in AB.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). The triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T of the band matrix A, stored in the first KD+1 rows of the array. The j-th column of U or L is stored in the j-th column of the array AB as follows: if UPLO ='U', AB(kd+1+i-j,j) = U(i,j) for max(1,j-kd)<=i<=j; if UPLO ='L', AB(1+i-j,j) = L(i,j) for j<=i<=min(n,j+kd).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

ANORM Input parameter.

ANORM is DOUBLE PRECISION

The 1-norm (or infinity-norm) of the symmetric band matrix A.

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cpbcon*, *spbcon* and *zpbcon*. It also exists with a native C interface as *LAPACKE_dpbcon*.

4.15.23 dpocon

`dpocon` estimates the reciprocal of the condition number (in the 1-norm) of a real symmetric positive definite matrix using the Cholesky factorization A = U^T *U or A = L*L^T computed by DPOTRF.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))).

Syntax

Fortran specification:

```
use armpl_library

subroutine dpocon(UPLO, N, A, LDA, ANORM, RCOND, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dpocon_(const char *uplo, const armpl_int_t *n, const double *a,
             const armpl_int_t *lda, const double *anorm, double *rcond,
             double *work, armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

= 'U': Upper triangle of `A` is stored; = 'L': Lower triangle of `A` is stored.

N Input parameter.

`N` is `INTEGER`

The order of the matrix `A`. `N` >= 0.

A Input parameter.

`A` is `DOUBLE PRECISION`

`A` is an array, dimension (`LDA`, `N`). The triangular factor U or L from the Cholesky factorization `A` = U^T *U or `A` = L*L^T , as computed by DPOTRF.

LDA Input parameter.

`LDA` is `INTEGER`

The leading dimension of the array `A`. `LDA` >= max(1, `N`).

ANORM Input parameter.

`ANORM` is `DOUBLE PRECISION`

The 1-norm (or infinity-norm) of the symmetric matrix `A`.

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cpocon*, *spocon* and *zpocon*. It also exists with a native C interface as *LAPACKE_dpocon*.

4.15.24 dppcon

dppcon estimates the reciprocal of the condition number (in the 1-norm) of a real symmetric positive definite packed matrix using the Cholesky factorization A = U^T *U or A = L*L^T computed by DPPTRF.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))).

Syntax

Fortran specification:

```
use armpl_library

subroutine dppcon(UPLO, N, AP, ANORM, RCOND, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dppcon_(const char *uplo, const armpl_int_t *n, const double *ap,
             const double *anorm, double *rcond, double *work,
             armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input parameter.

AP is DOUBLE PRECISION

AP is an array, dimension (N*(N+1)/2). The triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , packed columnwise in a linear array. The j-th column of U or L is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = U(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = L(i,j) for j<=i<=n.

ANORM Input parameter.

ANORM is DOUBLE PRECISION

The 1-norm (or infinity-norm) of the symmetric matrix A.

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cppcon*, *sppcon* and *zppcon*. It also exists with a native C interface as *LAPACKE_dppcon*.

4.15.25 dptcon

dptcon computes the reciprocal of the condition number (in the 1-norm) of a real symmetric positive definite tridiagonal matrix using the factorization A = L*D*L^T or A = U^T *D*U computed by DPTTRF.

Norm(inv(A)) is computed by a direct method, and the reciprocal of the condition number is computed as

```
RCOND = 1 / (ANORM * norm(inv(A))).
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dptcon(N, D, E, ANORM, RCOND, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dptcon_(const armpl_int_t *n, const double *d, const double *e,
             const double *anorm, double *rcond, double *work,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The n diagonal elements of the diagonal matrix D from the factorization of A, as computed by DPTTRF.

E Input parameter.

E is DOUBLE PRECISION

E is an array, dimension (N-1). The (n-1) off-diagonal elements of the unit bidiagonal factor U or L from the factorization of A, as computed by DPTTRF.

ANORM Input parameter.

ANORM is DOUBLE PRECISION

The 1-norm of the original matrix A.

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cptcon*, *sptcon* and *zptcon*. It also exists with a native C interface as *LAPACKE_dptcon*.

4.15.26 dspcon

dspcon estimates the reciprocal of the condition number (in the 1-norm) of a real symmetric packed matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by DSPTRF.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))).

Syntax

Fortran specification:

```
use armpl_library

subroutine dspcon(UPLO, N, AP, IPIV, ANORM, RCOND, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dspcon_(const char *uplo, const armpl_int_t *n, const double *ap,
             const armpl_int_t *ipiv, const double *anorm, double *rcond,
             double *work, armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input parameter.

AP is DOUBLE PRECISION

AP is an array, dimension (N*(N+1)/2). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by DSPTRF, stored as a packed triangular matrix.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by DSPTRF.

ANORM Input parameter.

ANORM is DOUBLE PRECISION

The 1-norm of the original matrix A.

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (2*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cspcon*, *sspcon* and *zspcon*. It also exists with a native C interface as *LAPACKE_dspcon*.

4.15.27 dsycon

dsycon estimates the reciprocal of the condition number (in the 1-norm) of a real symmetric matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by DSYTRF.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))).

Syntax

Fortran specification:

```
use armpl_library

subroutine dsycon(UPLO, N, A, LDA, IPIV, ANORM, RCOND, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dsycon_(const char *uplo, const armpl_int_t *n, const double *a,
             const armpl_int_t *lda, const armpl_int_t *ipiv,
             const double *anorm, double *rcond, double *work,
             armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by DSYTRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by DSYTRF.

ANORM Input parameter.

ANORM is DOUBLE PRECISION

The 1-norm of the original matrix A.

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (2*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csycon*, *ssycon* and *zsycon*. It also exists with a native C interface as *LAPACKE_dsycon*.

4.15.28 dsycon_3

DSYCON_3 estimates the reciprocal of the condition number (in the 1-norm) of a real symmetric matrix A using the factorization computed by DSYTRF_RK or DSYTRF_BK:

```
A = P*U*D*(U**T)*(P**T) or A = P*L*D*(L**T)*(P**T),
```

where U (or L) is unit upper (or lower) triangular matrix, U^T (or L^T) is the transpose of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))). This routine uses BLAS3 solver DSYTRS_3.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsycon_3(UPLO, N, A, LDA, E, IPIV, ANORM, RCOND, WORK, IWORK,
                    INFO)
```

C specification:

```
#include "armpl.h"

void dsycon_3_(const char *uplo, const armpl_int_t *n, const double *a,
               const armpl_int_t *lda, const double *e,
               const armpl_int_t *ipiv, const double *anorm, double *rcond,
               double *work, armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix: = 'U': Upper triangular, form is A = P*U*D*(U^T)*(P^T); = 'L': Lower triangular, form is A = P*L*D*(L^T)*(P^T).

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). Diagonal of the block diagonal matrix D and factors U or L as computed by DSYTRF_RK and DSYTRF_BK: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D should be provided on entry in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Input parameter.

E is DOUBLE PRECISION

E is an array, dimension (N). On entry, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i),i=2:N, E(1) not referenced; If UPLO = 'L': E(i) = D(i+1,i),i=1:N-1, E(N) not referenced.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is not referenced in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by DSYTRF_RK or DSYTRF_BK.

ANORM Input parameter.

ANORM is DOUBLE PRECISION

The 1-norm of the original matrix A.

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (2*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csycon_3*, *ssycon_3* and *zsycon_3*. It also exists with a native C interface as *LAPACKE_dsycon_3*.

4.15.29 dsycon_rook

DSYCON_ROOK estimates the reciprocal of the condition number (in the 1-norm) of a real symmetric matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by DSYTRF_ROOK.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))).

Syntax

Fortran specification:

```
use armpl_library

subroutine dsycon_rook(UPLO, N, A, LDA, IPIV, ANORM, RCOND, WORK, IWORK,
                       INFO)
```

C specification:

```
#include "armpl.h"

void dsycon_rook_(const char *uplo, const armpl_int_t *n, const double *a,
                  const armpl_int_t *lda, const armpl_int_t *ipiv,
                  const double *anorm, double *rcond, double *work,
                  armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by DSYTRF_ROOK.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by DSYTRF_ROOK.

ANORM Input parameter.

ANORM is DOUBLE PRECISION

The 1-norm of the original matrix A.

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (2*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csycon_rook*, *ssycon_rook* and *zsycon_rook*.

4.15.30 dtbcon

dtbcon estimates the reciprocal of the condition number of a triangular band matrix A, in either the 1-norm or the infinity-norm.

The norm of A is computed and an estimate is obtained for norm(inv(A)), then the reciprocal of the condition number is computed as

```
RCOND = 1 / ( norm(A) * norm(inv(A)) ).
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dtbcon(NORM, UPLO, DIAG, N, KD, AB, LDAB, RCOND, WORK, IWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dtbcon_(const char *norm, const char *uplo, const char *diag,
             const armpl_int_t *n, const armpl_int_t *kd, const double *ab,
             const armpl_int_t *ldab, double *rcond, double *work,
             armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies whether the 1-norm condition number or the infinity-norm condition number is required: = '1' or 'O': 1-norm; = 'I': Infinity-norm.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals or subdiagonals of the triangular band matrix A. KD >= 0.

AB Input parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). The upper or lower triangular band matrix A, stored in the first kd+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd). If DIAG = 'U', the diagonal elements of A are not referenced and are assumed to be 1.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(norm(A) * norm(inv(A))).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctbcon*, *stbcon* and *ztbcon*. It also exists with a native C interface as *LAPACKE_dtbcon*.

4.15.31 dtpcon

dtpcon estimates the reciprocal of the condition number of a packed triangular matrix A, in either the 1-norm or the infinity-norm.

The norm of A is computed and an estimate is obtained for norm(inv(A)), then the reciprocal of the condition number is computed as

```
RCOND = 1 / ( norm(A) * norm(inv(A)) ).
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dtpcon(NORM, UPLO, DIAG, N, AP, RCOND, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dtpcon_(const char *norm, const char *uplo, const char *diag,
             const armpl_int_t *n, const double *ap, double *rcond,
             double *work, armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies whether the 1-norm condition number or the infinity-norm condition number is required: = '1' or 'O': 1-norm; = 'I': Infinity-norm.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input parameter.

AP is DOUBLE PRECISION

AP is an array, dimension (N*(N+1)/2). The upper or lower triangular matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n. If DIAG = 'U', the diagonal elements of A are not referenced and are assumed to be 1.

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(norm(A) * norm(inv(A))).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctpcon*, *stpcon* and *ztpcon*. It also exists with a native C interface as *LAPACKE_dtpcon*.

4.15.32 dtrcon

dtrcon estimates the reciprocal of the condition number of a triangular matrix A, in either the 1-norm or the infinity-norm.

The norm of A is computed and an estimate is obtained for norm(inv(A)), then the reciprocal of the condition number is computed as

```
RCOND = 1 / ( norm(A) * norm(inv(A)) ).
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dtrcon(NORM, UPLO, DIAG, N, A, LDA, RCOND, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dtrcon_(const char *norm, const char *uplo, const char *diag,
             const armpl_int_t *n, const double *a, const armpl_int_t *lda,
             double *rcond, double *work, armpl_int_t *iwork,
             armpl_int_t *info, ... );
```

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies whether the 1-norm condition number or the infinity-norm condition number is required: = '1' or 'O': 1-norm; = 'I': Infinity-norm.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). The triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(norm(A) * norm(inv(A))).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctrcon*, *strcon* and *ztrcon*. It also exists with a native C interface as *LAPACKE_dtrcon*.

4.15.33 sgbcon

sgbcon estimates the reciprocal of the condition number of a real general band matrix A, in either the 1-norm or the infinity-norm, using the LU factorization computed by SGBTRF.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as

```
RCOND = 1 / ( norm(A) * norm(inv(A)) ).
```

Syntax

Fortran specification:

```
use armpl_library

subroutine sgbcon(NORM, N, KL, KU, AB, LDAB, IPIV, ANORM, RCOND, WORK, IWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void sgbcon_(const char *norm, const armpl_int_t *n, const armpl_int_t *kl,
             const armpl_int_t *ku, const float *ab, const armpl_int_t *ldab,
             const armpl_int_t *ipiv, const float *anorm, float *rcond,
             float *work, armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies whether the 1-norm condition number or the infinity-norm condition number is required: = ‘1’ or ‘O’: 1-norm; = ‘I’: Infinity-norm.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

AB Input parameter.

AB is REAL

AB is an array, dimension (LDAB, N). Details of the LU factorization of the band matrix A, as computed by SGBTRF. U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= 2*KL+KU+1.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices; for 1 <= i <= N, row i of the matrix was interchanged with row IPIV(i).

ANORM Input parameter.

ANORM is REAL

If NORM = '1' or 'O', the 1-norm of the original matrix A. If NORM = 'I', the infinity-norm of the original matrix A.

RCOND Output parameter.

RCOND is REAL

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(norm(A) * norm(inv(A))).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgbcon*, *dgbcon* and *zgbcon*. It also exists with a native C interface as *LAPACKE_sgbcon*.

4.15.34 sgecon

`sgecon` estimates the reciprocal of the condition number of a general real matrix A, in either the 1-norm or the infinity-norm, using the LU factorization computed by SGETRF.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as

```
RCOND = 1 / ( norm(A) * norm(inv(A)) ).
```

Syntax

Fortran specification:

```
use armpl_library

subroutine sgecon(NORM, N, A, LDA, ANORM, RCOND, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgecon_(const char *norm, const armpl_int_t *n, const float *a,
             const armpl_int_t *lda, const float *anorm, float *rcond,
             float *work, armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies whether the 1-norm condition number or the infinity-norm condition number is required: = '1' or 'O': 1-norm; = 'I': Infinity-norm.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). The factors L and U from the factorization A = P*L*U as computed by SGETRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

ANORM Input parameter.

ANORM is REAL

If NORM = '1' or 'O', the 1-norm of the original matrix A. If NORM = 'I', the infinity-norm of the original matrix A.

RCOND Output parameter.

RCOND is REAL

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(norm(A) * norm(inv(A))).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (4*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgecon*, *dgecon* and *zgecon*. It also exists with a native C interface as *LAPACKE_sgecon*.

4.15.35 sgtcon

sgtcon estimates the reciprocal of the condition number of a real tridiagonal matrix A using the LU factorization as computed by SGTTRF.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))).

Syntax

Fortran specification:

```
use armpl_library

subroutine sgtcon(NORM, N, DL, D, DU, DU2, IPIV, ANORM, RCOND, WORK, IWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void sgtcon_(const char *norm, const armpl_int_t *n, const float *dl,
             const float *d, const float *du, const float *du2,
             const armpl_int_t *ipiv, const float *anorm, float *rcond,
             float *work, armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies whether the 1-norm condition number or the infinity-norm condition number is required: = ‘1’ or ‘O’: 1-norm; = ‘I’: Infinity-norm.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

DL Input parameter.

DL is REAL

DL is an array, dimension (N-1). The (n-1) multipliers that define the matrix L from the LU factorization of A as computed by SGTTRF.

D Input parameter.

D is REAL

D is an array, dimension (N). The n diagonal elements of the upper triangular matrix U from the LU factorization of A.

DU Input parameter.

DU is REAL

DU is an array, dimension (N-1). The (n-1) elements of the first superdiagonal of U.

DU2 Input parameter.

DU2 is REAL

DU2 is an array, dimension (N-2). The (n-2) elements of the second superdiagonal of U.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices; for 1 <= i <= n, row i of the matrix was interchanged with row IPIV(i). IPIV(i) will always be either i or i+1; IPIV(i) = i indicates a row interchange was not required.

ANORM Input parameter.

ANORM is REAL

If NORM = '1' or 'O', the 1-norm of the original matrix A. If NORM = 'I', the infinity-norm of the original matrix A.

RCOND Output parameter.

RCOND is REAL

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (2*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgtcon*, *dgtcon* and *zgtcon*. It also exists with a native C interface as *LAPACKE_sgtcon*.

4.15.36 spbcon

spbcon estimates the reciprocal of the condition number (in the 1-norm) of a real symmetric positive definite band matrix using the Cholesky factorization A = U^T *U or A = L*L^T computed by SPBTRF.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))).

Syntax

Fortran specification:

```
use armpl_library

subroutine spbcon(UPLO, N, KD, AB, LDAB, ANORM, RCOND, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void spbcon_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
             const float *ab, const armpl_int_t *ldab, const float *anorm,
             float *rcond, float *work, armpl_int_t *iwork, armpl_int_t *info,
             ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangular factor stored in AB; = 'L': Lower triangular factor stored in AB.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input parameter.

AB is REAL

AB is an array, dimension (LDAB, N). The triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T of the band matrix A, stored in the first KD+1 rows of the array. The j-th column of U or L is stored in the j-th column of the array AB as follows: if UPLO ='U', AB(kd+1+i-j,j) = U(i,j) for max(1,j-kd)<=i<=j; if UPLO ='L', AB(1+i-j,j) = L(i,j) for j<=i<=min(n,j+kd).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

ANORM Input parameter.

ANORM is REAL

The 1-norm (or infinity-norm) of the symmetric band matrix A.

RCOND Output parameter.

RCOND is REAL

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cpbcon*, *dpbcon* and *zpbcon*. It also exists with a native C interface as *LAPACKE_spbcon*.

4.15.37 spocon

spocon estimates the reciprocal of the condition number (in the 1-norm) of a real symmetric positive definite matrix using the Cholesky factorization A = U^T *U or A = L*L^T computed by SPOTRF.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))).

Syntax

Fortran specification:

```
use armpl_library

subroutine spocon(UPLO, N, A, LDA, ANORM, RCOND, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void spocon_(const char *uplo, const armpl_int_t *n, const float *a,
             const armpl_int_t *lda, const float *anorm, float *rcond,
             float *work, armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). The triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , as computed by SPOTRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

ANORM Input parameter.

ANORM is REAL

The 1-norm (or infinity-norm) of the symmetric matrix A.

RCOND Output parameter.

RCOND is REAL

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cpocon*, *dpocon* and *zpocon*. It also exists with a native C interface as *LAPACKE_spocon*.

4.15.38 sppcon

sppcon estimates the reciprocal of the condition number (in the 1-norm) of a real symmetric positive definite packed matrix using the Cholesky factorization A = U^T *U or A = L*L^T computed by SPPTRF.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))).

Syntax

Fortran specification:

```
use armpl_library

subroutine sppcon(UPLO, N, AP, ANORM, RCOND, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sppcon_(const char *uplo, const armpl_int_t *n, const float *ap,
             const float *anorm, float *rcond, float *work,
             armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input parameter.

AP is REAL

AP is an array, dimension (N*(N+1)/2). The triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , packed columnwise in a linear array. The j-th column of U or L is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = U(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = L(i,j) for j<=i<=n.

ANORM Input parameter.

ANORM is REAL

The 1-norm (or infinity-norm) of the symmetric matrix A.

RCOND Output parameter.

RCOND is REAL

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cppcon*, *dppcon* and *zppcon*. It also exists with a native C interface as *LAPACKE_sppcon*.

4.15.39 sptcon

sptcon computes the reciprocal of the condition number (in the 1-norm) of a real symmetric positive definite tridiagonal matrix using the factorization A = L*D*L^T or A = U^T *D*U computed by SPTTRF.

Norm(inv(A)) is computed by a direct method, and the reciprocal of the condition number is computed as

```
RCOND = 1 / (ANORM * norm(inv(A))).
```

Syntax

Fortran specification:

```
use armpl_library

subroutine sptcon(N, D, E, ANORM, RCOND, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void sptcon_(const armpl_int_t *n, const float *d, const float *e,
             const float *anorm, float *rcond, float *work,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

D Input parameter.

D is REAL

D is an array, dimension (N). The n diagonal elements of the diagonal matrix D from the factorization of A, as computed by SPTTRF.

E Input parameter.

E is REAL

E is an array, dimension (N-1). The (n-1) off-diagonal elements of the unit bidiagonal factor U or L from the factorization of A, as computed by SPTTRF.

ANORM Input parameter.

ANORM is REAL

The 1-norm of the original matrix A.

RCOND Output parameter.

RCOND is REAL

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cptcon*, *dptcon* and *zptcon*. It also exists with a native C interface as *LAPACKE_sptcon*.

4.15.40 sspcon

sspcon estimates the reciprocal of the condition number (in the 1-norm) of a real symmetric packed matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by SSPTRF.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))).

Syntax

Fortran specification:

```
use armpl_library

subroutine sspcon(UPLO, N, AP, IPIV, ANORM, RCOND, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sspcon_(const char *uplo, const armpl_int_t *n, const float *ap,
             const armpl_int_t *ipiv, const float *anorm, float *rcond,
             float *work, armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input parameter.

AP is REAL

AP is an array, dimension (N*(N+1)/2). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by SSPTRF, stored as a packed triangular matrix.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by SSPTRF.

ANORM Input parameter.

ANORM is REAL

The 1-norm of the original matrix A.

RCOND Output parameter.

RCOND is REAL

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (2*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cspcon*, *dspcon* and *zspcon*. It also exists with a native C interface as *LAPACKE_sspcon*.

4.15.41 ssycon

ssycon estimates the reciprocal of the condition number (in the 1-norm) of a real symmetric matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by SSYTRF.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))).

Syntax

Fortran specification:

```
use armpl_library

subroutine ssycon(UPLO, N, A, LDA, IPIV, ANORM, RCOND, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ssycon_(const char *uplo, const armpl_int_t *n, const float *a,
             const armpl_int_t *lda, const armpl_int_t *ipiv,
             const float *anorm, float *rcond, float *work,
             armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by SSYTRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by SSYTRF.

ANORM Input parameter.

ANORM is REAL

The 1-norm of the original matrix A.

RCOND Output parameter.

RCOND is REAL

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (2*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csycon*, *dsycon* and *zsycon*. It also exists with a native C interface as *LAPACKE_ssycon*.

4.15.42 ssycon_3

SSYCON_3 estimates the reciprocal of the condition number (in the 1-norm) of a real symmetric matrix A using the factorization computed by DSYTRF_RK or DSYTRF_BK:

```
A = P*U*D*(U**T)*(P**T) or A = P*L*D*(L**T)*(P**T),
```

where U (or L) is unit upper (or lower) triangular matrix, U^T (or L^T) is the transpose of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))). This routine uses BLAS3 solver SSYTRS_3.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssycon_3(UPLO, N, A, LDA, E, IPIV, ANORM, RCOND, WORK, IWORK,
                    INFO)
```

C specification:

```
#include "armpl.h"

void ssycon_3_(const char *uplo, const armpl_int_t *n, const float *a,
               const armpl_int_t *lda, const float *e,
               const armpl_int_t *ipiv, const float *anorm, float *rcond,
               float *work, armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix: = 'U': Upper triangular, form is A = P*U*D*(U^T)*(P^T); = 'L': Lower triangular, form is A = P*L*D*(L^T)*(P^T).

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). Diagonal of the block diagonal matrix D and factors U or L as computed by SSYTRF_RK and SSYTRF_BK: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D should be provided on entry in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Input parameter.

E is REAL

E is an array, dimension (N). On entry, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i),i=2:N, E(1) not referenced; If UPLO = 'L': E(i) = D(i+1,i),i=1:N-1, E(N) not referenced.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is not referenced in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by SSYTRF_RK or SSYTRF_BK.

ANORM Input parameter.

ANORM is REAL

The 1-norm of the original matrix A.

RCOND Output parameter.

RCOND is REAL

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (2*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csycon_3*, *dsycon_3* and *zsycon_3*. It also exists with a native C interface as *LAPACKE_ssycon_3*.

4.15.43 ssycon_rook

SSYCON_ROOK estimates the reciprocal of the condition number (in the 1-norm) of a real symmetric matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by SSYTRF_ROOK.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))).

Syntax

Fortran specification:

```
use armpl_library

subroutine ssycon_rook(UPLO, N, A, LDA, IPIV, ANORM, RCOND, WORK, IWORK,
                       INFO)
```

C specification:

```
#include "armpl.h"

void ssycon_rook_(const char *uplo, const armpl_int_t *n, const float *a,
                  const armpl_int_t *lda, const armpl_int_t *ipiv,
                  const float *anorm, float *rcond, float *work,
                  armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by SSYTRF_ROOK.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by SSYTRF_ROOK.

ANORM Input parameter.

ANORM is REAL

The 1-norm of the original matrix A.

RCOND Output parameter.

RCOND is REAL

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (2*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csycon_rook*, *dsycon_rook* and *zsycon_rook*.

4.15.44 stbcon

`stbcon` estimates the reciprocal of the condition number of a triangular band matrix A, in either the 1-norm or the infinity-norm.

The norm of A is computed and an estimate is obtained for norm(inv(A)), then the reciprocal of the condition number is computed as

```
RCOND = 1 / ( norm(A) * norm(inv(A)) ).
```

Syntax

Fortran specification:

```
use armpl_library

subroutine stbcon(NORM, UPLO, DIAG, N, KD, AB, LDAB, RCOND, WORK, IWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void stbcon_(const char *norm, const char *uplo, const char *diag,
             const armpl_int_t *n, const armpl_int_t *kd, const float *ab,
             const armpl_int_t *ldab, float *rcond, float *work,
             armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

NORM Input parameter.

`NORM` is `CHARACTER*1`

Specifies whether the 1-norm condition number or the infinity-norm condition number is required: = '1' or 'O': 1-norm; = 'I': Infinity-norm.

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

= '`U`': A is upper triangular; = '`L`': A is lower triangular.

DIAG Input parameter.

`DIAG` is `CHARACTER*1`

= '`N`': A is non-unit triangular; = '`U`': A is unit triangular.

N Input parameter.

`N` is `INTEGER`

The order of the matrix A. `N` >= 0.

KD Input parameter.

`KD` is `INTEGER`

The number of superdiagonals or subdiagonals of the triangular band matrix A. `KD` >= 0.

AB Input parameter.

AB is REAL

AB is an array, dimension (LDAB, N). The upper or lower triangular band matrix A, stored in the first kd+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd). If DIAG = 'U', the diagonal elements of A are not referenced and are assumed to be 1.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

RCOND Output parameter.

RCOND is REAL

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(norm(A) * norm(inv(A))).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctbcon*, *dtbcon* and *ztbcon*. It also exists with a native C interface as *LAPACKE_stbcon*.

4.15.45 stpcon

stpcon estimates the reciprocal of the condition number of a packed triangular matrix A, in either the 1-norm or the infinity-norm.

The norm of A is computed and an estimate is obtained for norm(inv(A)), then the reciprocal of the condition number is computed as

```
RCOND = 1 / ( norm(A) * norm(inv(A)) ).
```

Syntax

Fortran specification:

```
use armpl_library

subroutine stpcon(NORM, UPLO, DIAG, N, AP, RCOND, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void stpcon_(const char *norm, const char *uplo, const char *diag,
             const armpl_int_t *n, const float *ap, float *rcond, float *work,
             armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies whether the 1-norm condition number or the infinity-norm condition number is required: = '1' or 'O': 1-norm; = 'I': Infinity-norm.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input parameter.

AP is REAL

AP is an array, dimension (N*(N+1)/2). The upper or lower triangular matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n. If DIAG = 'U', the diagonal elements of A are not referenced and are assumed to be 1.

RCOND Output parameter.

RCOND is REAL

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(norm(A) * norm(inv(A))).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctpcon*, *dtpcon* and *ztpcon*. It also exists with a native C interface as *LAPACKE_stpcon*.

4.15.46 strcon

`strcon` estimates the reciprocal of the condition number of a triangular matrix A, in either the 1-norm or the infinity-norm.

The norm of A is computed and an estimate is obtained for norm(inv(A)), then the reciprocal of the condition number is computed as

```
RCOND = 1 / ( norm(A) * norm(inv(A)) ).
```

Syntax

Fortran specification:

```
use armpl_library

subroutine strcon(NORM, UPLO, DIAG, N, A, LDA, RCOND, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void strcon_(const char *norm, const char *uplo, const char *diag,
             const armpl_int_t *n, const float *a, const armpl_int_t *lda,
             float *rcond, float *work, armpl_int_t *iwork, armpl_int_t *info,
             ... );
```

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies whether the 1-norm condition number or the infinity-norm condition number is required: = '1' or 'O': 1-norm; = 'I': Infinity-norm.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). The triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

RCOND Output parameter.

RCOND is REAL

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(norm(A) * norm(inv(A))).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (3*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctrcon*, *dtrcon* and *ztrcon*. It also exists with a native C interface as *LAPACKE_strcon*.

4.15.47 zgbcon

zgbcon estimates the reciprocal of the condition number of a complex general band matrix A, in either the 1-norm or the infinity-norm, using the LU factorization computed by ZGBTRF.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as

```
RCOND = 1 / ( norm(A) * norm(inv(A)) ).
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zgbcon(NORM, N, KL, KU, AB, LDAB, IPIV, ANORM, RCOND, WORK, RWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void zgbcon_(const char *norm, const armpl_int_t *n, const armpl_int_t *kl,
             const armpl_int_t *ku, const armpl_doublecomplex_t *ab,
             const armpl_int_t *ldab, const armpl_int_t *ipiv,
             const double *anorm, double *rcond, armpl_doublecomplex_t *work,
             double *rwork, armpl_int_t *info, ... );
```

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies whether the 1-norm condition number or the infinity-norm condition number is required: = '1' or 'O': 1-norm; = 'I': Infinity-norm.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

AB Input parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). Details of the LU factorization of the band matrix A, as computed by ZGBTRF. U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= 2*KL+KU+1.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices; for 1 <= i <= N, row i of the matrix was interchanged with row IPIV(i).

ANORM Input parameter.

ANORM is DOUBLE PRECISION

If NORM = '1' or 'O', the 1-norm of the original matrix A. If NORM = 'I', the infinity-norm of the original matrix A.

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(norm(A) * norm(inv(A))).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgbcon*, *dgbcon* and *sgbcon*. It also exists with a native C interface as *LAPACKE_zgbcon*.

4.15.48 zgecon

zgecon estimates the reciprocal of the condition number of a general complex matrix A, in either the 1-norm or the infinity-norm, using the LU factorization computed by ZGETRF.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as

```
RCOND = 1 / ( norm(A) * norm(inv(A)) ).
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zgecon(NORM, N, A, LDA, ANORM, RCOND, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgecon_(const char *norm, const armpl_int_t *n,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const double *anorm, double *rcond, armpl_doublecomplex_t *work,
             double *rwork, armpl_int_t *info, ... );
```

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies whether the 1-norm condition number or the infinity-norm condition number is required: = '1' or 'O': 1-norm; = 'I': Infinity-norm.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The factors L and U from the factorization A = P*L*U as computed by ZGETRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

ANORM Input parameter.

ANORM is DOUBLE PRECISION

If NORM = '1' or 'O', the 1-norm of the original matrix A. If NORM = 'I', the infinity-norm of the original matrix A.

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(norm(A) * norm(inv(A))).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgecon*, *dgecon* and *sgecon*. It also exists with a native C interface as *LAPACKE_zgecon*.

4.15.49 zgtcon

zgtcon estimates the reciprocal of the condition number of a complex tridiagonal matrix A using the LU factorization as computed by ZGTTRF.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))).

Syntax

Fortran specification:

```
use armpl_library

subroutine zgtcon(NORM, N, DL, D, DU, DU2, IPIV, ANORM, RCOND, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgtcon_(const char *norm, const armpl_int_t *n,
             const armpl_doublecomplex_t *dl, const armpl_doublecomplex_t *d,
             const armpl_doublecomplex_t *du,
```

(continues on next page)

(continued from previous page)

```
                const armpl_doublecomplex_t *du2, const armpl_int_t *ipiv,
                const double *anorm, double *rcond, armpl_doublecomplex_t *work,
                armpl_int_t *info, ... );
```

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies whether the 1-norm condition number or the infinity-norm condition number is required: = '1' or 'O': 1-norm; = 'I': Infinity-norm.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

DL Input parameter.

DL is COMPLEX*16

DL is an array, dimension (N-1). The (n-1) multipliers that define the matrix L from the LU factorization of A as computed by ZGTTRF.

D Input parameter.

D is COMPLEX*16

D is an array, dimension (N). The n diagonal elements of the upper triangular matrix U from the LU factorization of A.

DU Input parameter.

DU is COMPLEX*16

DU is an array, dimension (N-1). The (n-1) elements of the first superdiagonal of U.

DU2 Input parameter.

DU2 is COMPLEX*16

DU2 is an array, dimension (N-2). The (n-2) elements of the second superdiagonal of U.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices; for 1 <= i <= n, row i of the matrix was interchanged with row IPIV(i). IPIV(i) will always be either i or i+1; IPIV(i) = i indicates a row interchange was not required.

ANORM Input parameter.

ANORM is DOUBLE PRECISION

If NORM = '1' or 'O', the 1-norm of the original matrix A. If NORM = 'I', the infinity-norm of the original matrix A.

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgtcon*, *dgtcon* and *sgtcon*. It also exists with a native C interface as *LAPACKE_zgtcon*.

4.15.50 zhecon

zhecon estimates the reciprocal of the condition number of a complex Hermitian matrix A using the factorization A = U*D*U^H or A = L*D*L^H computed by ZHETRF.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))).

Syntax

Fortran specification:

```
use armpl_library

subroutine zhecon(UPLO, N, A, LDA, IPIV, ANORM, RCOND, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhecon_(const char *uplo, const armpl_int_t *n,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_int_t *ipiv, const double *anorm, double *rcond,
             armpl_doublecomplex_t *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^H ; = 'L': Lower triangular, form is A = L*D*L^H .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by ZHETRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZHETRF.

ANORM Input parameter.

ANORM is DOUBLE PRECISION

The 1-norm of the original matrix A.

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *checon*. It also exists with a native C interface as *LAPACKE_zhecon*.

4.15.51 zhecon_3

ZHECON_3 estimates the reciprocal of the condition number (in the 1-norm) of a complex Hermitian matrix A using the factorization computed by ZHETRF_RK or ZHETRF_BK:

```
A = P*U*D*(U**H)*(P**T) or A = P*L*D*(L**H)*(P**T),
```

where U (or L) is unit upper (or lower) triangular matrix, U^H (or L^H) is the conjugate of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is Hermitian and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))). This routine uses BLAS3 solver ZHETRS_3.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhecon_3(UPLO, N, A, LDA, E, IPIV, ANORM, RCOND, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhecon_3_(const char *uplo, const armpl_int_t *n,
               const armpl_doublecomplex_t *a, const armpl_int_t *lda,
               const armpl_doublecomplex_t *e, const armpl_int_t *ipiv,
               const double *anorm, double *rcond,
               armpl_doublecomplex_t *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix: = 'U': Upper triangular, form is A = P*U*D*(U^H)*(P^T); = 'L': Lower triangular, form is A = P*L*D*(L^H)*(P^T).

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). Diagonal of the block diagonal matrix D and factors U or L as computed by ZHETRF_RK and ZHETRF_BK: a) ONLY diagonal elements of the Hermitian block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D should be provided on entry in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Input parameter.

E is COMPLEX*16

E is an array, dimension (N). On entry, contains the superdiagonal (or subdiagonal) elements of the Hermitian block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i),i=2:N, E(1) not referenced; If UPLO = 'L': E(i) = D(i+1,i),i=1:N-1, E(N) not referenced.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is not referenced in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZHETRF_RK or ZHETRF_BK.

ANORM Input parameter.

ANORM is DOUBLE PRECISION

The 1-norm of the original matrix A.

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *checon_3*. It also exists with a native C interface as *LAPACKE_zhecon_3*.

4.15.52 zhecon_rook

ZHECON_ROOK estimates the reciprocal of the condition number of a complex Hermitian matrix A using the factorization A = U*D*U^H or A = L*D*L^H computed by CHETRF_ROOK.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))).

Syntax

Fortran specification:

```
use armpl_library

subroutine zhecon_rook(UPLO, N, A, LDA, IPIV, ANORM, RCOND, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhecon_rook_(const char *uplo, const armpl_int_t *n,
                  const armpl_doublecomplex_t *a, const armpl_int_t *lda,
                  const armpl_int_t *ipiv, const double *anorm, double *rcond,
                  armpl_doublecomplex_t *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^H ; = 'L': Lower triangular, form is A = L*D*L^H .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CHETRF_ROOK.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CHETRF_ROOK.

ANORM Input parameter.

ANORM is DOUBLE PRECISION

The 1-norm of the original matrix A.

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *checon_rook*.

4.15.53 zhpcon

zhpcon estimates the reciprocal of the condition number of a complex Hermitian packed matrix A using the factorization A = U*D*U^H or A = L*D*L^H computed by ZHPTRF.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))).

Syntax

Fortran specification:

```
use armpl_library

subroutine zhpcon(UPLO, N, AP, IPIV, ANORM, RCOND, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhpcon_(const char *uplo, const armpl_int_t *n,
             const armpl_doublecomplex_t *ap, const armpl_int_t *ipiv,
             const double *anorm, double *rcond, armpl_doublecomplex_t *work,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^H ; = 'L': Lower triangular, form is A = L*D*L^H .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by ZHPTRF, stored as a packed triangular matrix.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZHPTRF.

ANORM Input parameter.

ANORM is DOUBLE PRECISION

The 1-norm of the original matrix A.

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *chpcon*. It also exists with a native C interface as *LAPACKE_zhpcon*.

4.15.54 zpbcon

zpbcon estimates the reciprocal of the condition number (in the 1-norm) of a complex Hermitian positive definite band matrix using the Cholesky factorization A = U^H *U or A = L*L^H computed by ZPBTRF.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))).

Syntax

Fortran specification:

```
use armpl_library

subroutine zpbcon(UPLO, N, KD, AB, LDAB, ANORM, RCOND, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zpbcon_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
             const armpl_doublecomplex_t *ab, const armpl_int_t *ldab,
             const double *anorm, double *rcond, armpl_doublecomplex_t *work,
             double *rwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangular factor stored in AB; = 'L': Lower triangular factor stored in AB.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of sub-diagonals if UPLO = 'L'. KD >= 0.

AB Input parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). The triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H of the band matrix A, stored in the first KD+1 rows of the array. The j-th column of U or L is stored in the j-th column of the array AB as follows: if UPLO ='U', AB(kd+1+i-j,j) = U(i,j) for max(1,j-kd)<=i<=j; if UPLO ='L', AB(1+i-j,j) = L(i,j) for j<=i<=min(n,j+kd).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

ANORM Input parameter.

ANORM is DOUBLE PRECISION

The 1-norm (or infinity-norm) of the Hermitian band matrix A.

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cpbcon*, *dpbcon* and *spbcon*. It also exists with a native C interface as *LAPACKE_zpbcon*.

4.15.55 zpocon

zpocon estimates the reciprocal of the condition number (in the 1-norm) of a complex Hermitian positive definite matrix using the Cholesky factorization A = U^H *U or A = L*L^H computed by ZPOTRF.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))).

Syntax

Fortran specification:

```
use armpl_library

subroutine zpocon(UPLO, N, A, LDA, ANORM, RCOND, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zpocon_(const char *uplo, const armpl_int_t *n,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const double *anorm, double *rcond, armpl_doublecomplex_t *work,
             double *rwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H , as computed by ZPOTRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

ANORM Input parameter.

ANORM is DOUBLE PRECISION

The 1-norm (or infinity-norm) of the Hermitian matrix A.

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cpocon*, *dpocon* and *spocon*. It also exists with a native C interface as *LAPACKE_zpocon*.

4.15.56 zppcon

zppcon estimates the reciprocal of the condition number (in the 1-norm) of a complex Hermitian positive definite packed matrix using the Cholesky factorization A = U^H *U or A = L*L^H computed by ZPPTRF.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))).

Syntax

Fortran specification:

```
use armpl_library

subroutine zppcon(UPLO, N, AP, ANORM, RCOND, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zppcon_(const char *uplo, const armpl_int_t *n,
             const armpl_doublecomplex_t *ap, const double *anorm,
             double *rcond, armpl_doublecomplex_t *work, double *rwork,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). The triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H , packed columnwise in a linear array. The j-th column of U or L is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = U(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = L(i,j) for j<=i<=n.

ANORM Input parameter.

ANORM is DOUBLE PRECISION

The 1-norm (or infinity-norm) of the Hermitian matrix A.

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cppcon*, *dppcon* and *sppcon*. It also exists with a native C interface as *LAPACKE_zppcon*.

4.15.57 zptcon

`zptcon` computes the reciprocal of the condition number (in the 1-norm) of a complex Hermitian positive definite tridiagonal matrix using the factorization A = L*D*L^H or A = U^H *D*U computed by ZPTTRF.

Norm(inv(A)) is computed by a direct method, and the reciprocal of the condition number is computed as

```
RCOND = 1 / (ANORM * norm(inv(A))).
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zptcon(N, D, E, ANORM, RCOND, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zptcon_(const armpl_int_t *n, const double *d,
             const armpl_doublecomplex_t *e, const double *anorm,
             double *rcond, double *rwork, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The n diagonal elements of the diagonal matrix D from the factorization of A, as computed by ZPTTRF.

E Input parameter.

E is COMPLEX*16

E is an array, dimension (N-1). The (n-1) off-diagonal elements of the unit bidiagonal factor U or L from the factorization of A, as computed by ZPTTRF.

ANORM Input parameter.

ANORM is DOUBLE PRECISION

The 1-norm of the original matrix A.

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is the 1-norm of inv(A) computed in this routine.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cptcon*, *dptcon* and *sptcon*. It also exists with a native C interface as *LAPACKE_zptcon*.

4.15.58 zspcon

zspcon estimates the reciprocal of the condition number (in the 1-norm) of a complex symmetric packed matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by ZSPTRF.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))).

Syntax

Fortran specification:

```
use armpl_library

subroutine zspcon(UPLO, N, AP, IPIV, ANORM, RCOND, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zspcon_(const char *uplo, const armpl_int_t *n,
             const armpl_doublecomplex_t *ap, const armpl_int_t *ipiv,
             const double *anorm, double *rcond, armpl_doublecomplex_t *work,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by ZSPTRF, stored as a packed triangular matrix.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZSPTRF.

ANORM Input parameter.

ANORM is DOUBLE PRECISION

The 1-norm of the original matrix A.

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cspcon*, *dspcon* and *sspcon*. It also exists with a native C interface as *LAPACKE_zspcon*.

4.15.59 zsycon

zsycon estimates the reciprocal of the condition number (in the 1-norm) of a complex symmetric matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by ZSYTRF.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))).

Syntax

Fortran specification:

```
use armpl_library

subroutine zsycon(UPLO, N, A, LDA, IPIV, ANORM, RCOND, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zsycon_(const char *uplo, const armpl_int_t *n,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_int_t *ipiv, const double *anorm, double *rcond,
             armpl_doublecomplex_t *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by ZSYTRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZSYTRF.

ANORM Input parameter.

ANORM is DOUBLE PRECISION

The 1-norm of the original matrix A.

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csycon*, *dsycon* and *ssycon*. It also exists with a native C interface as *LAPACKE_zsycon*.

4.15.60 zsycon_3

ZSYCON_3 estimates the reciprocal of the condition number (in the 1-norm) of a complex symmetric matrix A using the factorization computed by ZSYTRF_RK or ZSYTRF_BK:

```
A = P*U*D*(U**T)*(P**T) or A = P*L*D*(L**T)*(P**T),
```

where U (or L) is unit upper (or lower) triangular matrix, U^T (or L^T) is the transpose of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))). This routine uses BLAS3 solver ZSYTRS_3.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsycon_3(UPLO, N, A, LDA, E, IPIV, ANORM, RCOND, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zsycon_3_(const char *uplo, const armpl_int_t *n,
               const armpl_doublecomplex_t *a, const armpl_int_t *lda,
               const armpl_doublecomplex_t *e, const armpl_int_t *ipiv,
               const double *anorm, double *rcond,
               const armpl_doublecomplex_t *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix: = 'U': Upper triangular, form is A = P*U*D*(U^T)*(P^T); = 'L': Lower triangular, form is A = P*L*D*(L^T)*(P^T).

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). Diagonal of the block diagonal matrix D and factors U or L as computed by ZSYTRF_RK and ZSYTRF_BK: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D should be provided on entry in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Input parameter.

E is COMPLEX*16

E is an array, dimension (N). On entry, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i),i=2:N, E(1) not referenced; If UPLO = 'L': E(i) = D(i+1,i),i=1:N-1, E(N) not referenced.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is not referenced in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZSYTRF_RK or ZSYTRF_BK.

ANORM Input parameter.

ANORM is DOUBLE PRECISION

The 1-norm of the original matrix A.

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csycon_3*, *dsycon_3* and *ssycon_3*. It also exists with a native C interface as *LAPACKE_zsycon_3*.

4.15.61 zsycon_rook

ZSYCON_ROOK estimates the reciprocal of the condition number (in the 1-norm) of a complex symmetric matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by ZSYTRF_ROOK.

An estimate is obtained for norm(inv(A)), and the reciprocal of the condition number is computed as RCOND = 1 / (ANORM * norm(inv(A))).

Syntax

Fortran specification:

```
use armpl_library

subroutine zsycon_rook(UPLO, N, A, LDA, IPIV, ANORM, RCOND, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zsycon_rook_(const char *uplo, const armpl_int_t *n,
                  const armpl_doublecomplex_t *a, const armpl_int_t *lda,
```

(continues on next page)

(continued from previous page)

```
                    const armpl_int_t *ipiv, const double *anorm, double *rcond,
                    armpl_doublecomplex_t *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by ZSYTRF_ROOK.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZSYTRF_ROOK.

ANORM Input parameter.

ANORM is DOUBLE PRECISION

The 1-norm of the original matrix A.

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(ANORM * AINVNM), where AINVNM is an estimate of the 1-norm of inv(A) computed in this routine.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csycon_rook*, *dsycon_rook* and *ssycon_rook*.

4.15.62 ztbcon

ztbcon estimates the reciprocal of the condition number of a triangular band matrix A, in either the 1-norm or the infinity-norm.

The norm of A is computed and an estimate is obtained for norm(inv(A)), then the reciprocal of the condition number is computed as

```
RCOND = 1 / ( norm(A) * norm(inv(A)) ).
```

Syntax

Fortran specification:

```
use armpl_library

subroutine ztbcon(NORM, UPLO, DIAG, N, KD, AB, LDAB, RCOND, WORK, RWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void ztbcon_(const char *norm, const char *uplo, const char *diag,
             const armpl_int_t *n, const armpl_int_t *kd,
             const armpl_doublecomplex_t *ab, const armpl_int_t *ldab,
             double *rcond, armpl_doublecomplex_t *work, double *rwork,
             armpl_int_t *info, ... );
```

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies whether the 1-norm condition number or the infinity-norm condition number is required: = '1' or 'O': 1-norm; = 'I': Infinity-norm.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals or subdiagonals of the triangular band matrix A. KD >= 0.

AB Input parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). The upper or lower triangular band matrix A, stored in the first kd+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd). If DIAG = 'U', the diagonal elements of A are not referenced and are assumed to be 1.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(norm(A) * norm(inv(A))).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctbcon*, *dtbcon* and *stbcon*. It also exists with a native C interface as *LAPACKE_ztbcon*.

4.15.63 ztpcon

ztpcon estimates the reciprocal of the condition number of a packed triangular matrix A, in either the 1-norm or the infinity-norm.

The norm of A is computed and an estimate is obtained for norm(inv(A)), then the reciprocal of the condition number is computed as

```
RCOND = 1 / ( norm(A) * norm(inv(A)) ).
```

Syntax

Fortran specification:

```
use armpl_library

subroutine ztpcon(NORM, UPLO, DIAG, N, AP, RCOND, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ztpcon_(const char *norm, const char *uplo, const char *diag,
             const armpl_int_t *n, const armpl_doublecomplex_t *ap,
             double *rcond, armpl_doublecomplex_t *work, double *rwork,
             armpl_int_t *info, ... );
```

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies whether the 1-norm condition number or the infinity-norm condition number is required: = '1' or 'O': 1-norm; = 'I': Infinity-norm.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). The upper or lower triangular matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n. If DIAG = 'U', the diagonal elements of A are not referenced and are assumed to be 1.

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(norm(A) * norm(inv(A))).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctpcon*, *dtpcon* and *stpcon*. It also exists with a native C interface as *LAPACKE_ztpcon*.

4.15.64 ztrcon

`ztrcon` estimates the reciprocal of the condition number of a triangular matrix A, in either the 1-norm or the infinity-norm.

The norm of A is computed and an estimate is obtained for norm(inv(A)), then the reciprocal of the condition number is computed as

```
RCOND = 1 / ( norm(A) * norm(inv(A)) ).
```

Syntax

Fortran specification:

```
use armpl_library

subroutine ztrcon(NORM, UPLO, DIAG, N, A, LDA, RCOND, WORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ztrcon_(const char *norm, const char *uplo, const char *diag,
             const armpl_int_t *n, const armpl_doublecomplex_t *a,
             const armpl_int_t *lda, double *rcond,
             armpl_doublecomplex_t *work, double *rwork, armpl_int_t *info,
             ... );
```

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies whether the 1-norm condition number or the infinity-norm condition number is required: = '1' or 'O': 1-norm; = 'I': Infinity-norm.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

RCOND Output parameter.

RCOND is DOUBLE PRECISION

The reciprocal of the condition number of the matrix A, computed as RCOND = 1/(norm(A) * norm(inv(A))).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctrcon*, *dtrcon* and *strcon*. It also exists with a native C interface as *LAPACKE_ztrcon*.

4.16 LAPACK lapack routines routines

4.16.1 cgetsls

cgetsls solves overdetermined or underdetermined complex linear systems involving an M-by-N matrix A, using a tall sk

The following options are provided:

* If TRANS = 'N' and m >= n: find the least squares solution of an overdetermined system, i.e., solve the least squares problem minimize || B - A*X ||.

* If TRANS = 'N' and m < n: find the minimum norm solution of an underdetermined system A * X = B.

* If TRANS = 'C' and m >= n: find the minimum norm solution of an undetermined system A^T * X = B.

* If TRANS = 'C' and m < n: find the least squares solution of an overdetermined system, i.e., solve the least squares problem minimize || B - A^T * X ||.

Several right hand side vectors b and solution vectors x can be handled in a single call; they are stored as the columns of the M-by-NRHS right hand side matrix B and the N-by-NRHS solution matrix X.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgetsls(TRANS, M, N, NRHS, A, LDA, B, LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgetsls_(const char *trans, const armpl_int_t *m, const armpl_int_t *n,
              const armpl_int_t *nrhs, armpl_singlecomplex_t *a,
              const armpl_int_t *lda, armpl_singlecomplex_t *b,
              const armpl_int_t *ldb, armpl_singlecomplex_t *work,
              const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': the linear system involves A; = 'C': the linear system involves A^H .

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >=0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, A is overwritten by details of its QR or LQ factorization as returned by CGEQR or CGELQ.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the matrix B of right hand side vectors, stored columnwise; B is M-by-NRHS if TRANS = 'N', or N-by-NRHS if TRANS = 'C'. On exit, if INFO = 0, B is overwritten by the solution vectors, stored columnwise: if TRANS = 'N' and m >= n, rows 1 to n of B contain the least squares solution vectors. if TRANS = 'N' and m < n, rows 1 to N of B contain the minimum norm solution vectors; if TRANS = 'C' and m >= n, rows 1 to M of B contain the minimum norm solution vectors; if TRANS = 'C' and m < n, rows 1 to M of B contain the least squares solution vectors.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= MAX(1, M, N).

WORK Output parameter.

(workspace) COMPLEX

(workspace) is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) contains optimal (or either minimal or optimal, if query was assumed) LWORK. See LWORK for details.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If LWORK = -1 or -2, then a workspace query is assumed. If LWORK = -1, the routine calculates optimal size of WORK for the optimal performance and returns this value in WORK(1). If LWORK = -2, the routine calculates minimal size of WORK and returns this value in WORK(1).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element of the triangular factor of A is zero, so that A does not have full rank; the least squares solution could not be computed.

Related Information

For this routine in other precisions, please see *dgetsls*, *sgetsls* and *zgetsls*. It also exists with a native C interface as *LAPACKE_cgetsls*.

4.16.2 chb2st_kernels

CHB2ST_KERNELS is an internal routine used by the CHETRD_HB2ST subroutine.

Syntax

Fortran specification:

```
use armpl_library

subroutine chb2st_kernels(UPLO, WANTZ, TTYPE, ST, ED, SWEEP, N, NB, IB, A,
                          LDA, V, TAU, LDVT, WORK)
```

C specification:

```
#include "armpl.h"

void chb2st_kernels_(const char *uplo, const armpl_int_t *wantz,
                     const armpl_int_t *ttype, const armpl_int_t *st,
                     const armpl_int_t *ed, const armpl_int_t *sweep,
                     const armpl_int_t *n, const armpl_int_t *nb,
                     const armpl_int_t *ib, armpl_singlecomplex_t *a,
                     const armpl_int_t *lda, armpl_singlecomplex_t *v,
                     armpl_singlecomplex_t *tau, const armpl_int_t *ldvt,
                     const armpl_singlecomplex_t *work, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

WANTZ Input parameter.

WANTZ is LOGICAL which indicate if Eigenvalue are requested or both

Eigenvalue/Eigenvectors.

TTYPE Input parameter.

TTYPE is INTEGER

ST Input parameter.

ST is INTEGER

internal parameter for indices.

ED Input parameter.

ED is INTEGER

internal parameter for indices.

SWEEP Input parameter.

SWEEP is INTEGER

internal parameter for indices.

N Input parameter.

N is INTEGER. The order of the matrix A.

NB Input parameter.

NB is INTEGER. The size of the band.

IB Input parameter.

IB is INTEGER.

A Input and output parameter.

A is COMPLEX array. A pointer to the matrix A.

LDA Input parameter.

LDA is INTEGER. The leading dimension of the matrix A.

V Output parameter.

V is COMPLEX

V is an array, dimension 2*n if eigenvalues only are. requested or to be queried for vectors.

TAU Output parameter.

TAU is COMPLEX

TAU is an array, dimension (2*n).. The scalar factors of the Householder reflectors are stored in this array.

LDVT Input parameter.

LDVT is INTEGER.

WORK Input parameter.

WORK is COMPLEX array. Workspace of size nb.

Related Information

For this routine in other precisions, please see *zhb2st_kernels*.

4.16.3 chbev_2stage

CHBEV_2STAGE computes all the eigenvalues and, optionally, eigenvectors of a complex Hermitian band matrix A using the 2stage technique for the reduction to tridiagonal.

Syntax

Fortran specification:

```
use armpl_library

subroutine chbev_2stage(JOBZ, UPLO, N, KD, AB, LDAB, W, Z, LDZ, WORK, LWORK,
                        RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void chbev_2stage_(const char *jobz, const char *uplo, const armpl_int_t *n,
                   const armpl_int_t *kd, armpl_singlecomplex_t *ab,
                   const armpl_int_t *ldab, float *w,
                   armpl_singlecomplex_t *z, const armpl_int_t *ldz,
                   armpl_singlecomplex_t *work, const armpl_int_t *lwork,
                   float *rwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors. Not available in this release.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the Hermitian band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, AB is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the first superdiagonal and the diagonal of the tridiagonal matrix T are returned in rows KD and KD+1 of AB, and if UPLO = 'L', the diagonal and first subdiagonal of T are returned in the first two rows of AB.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD + 1.

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the orthonormal eigenvectors of the matrix A, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension LWORK. On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= 1, when N <= 1; otherwise If JOBZ = 'N' and N > 1, LWORK must be queried. LWORK = MAX(1, dimension) where dimension = (2KD+1)* N + KD*NTHREADS where KD is the size of the band. NTHREADS is the number of threads used when openMP compilation is enabled, otherwise =1. If JOBZ = 'V' and N > 1, LWORK must be queried. Not yet available.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (max(1,3*N-2)) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *zhbev_2stage*. It also exists with a native C interface as *LAPACKE_chbev_2stage*.

4.16.4 chbevd_2stage

CHBEVD_2STAGE computes all the eigenvalues and, optionally, eigenvectors of a complex Hermitian band matrix A using the 2stage technique for the reduction to tridiagonal. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine chbevd_2stage(JOBZ, UPLO, N, KD, AB, LDAB, W, Z, LDZ, WORK, LWORK,
                         RWORK, LRWORK, IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void chbevd_2stage_(const char *jobz, const char *uplo, const armpl_int_t *n,
                    const armpl_int_t *kd, armpl_singlecomplex_t *ab,
                    const armpl_int_t *ldab, float *w,
                    armpl_singlecomplex_t *z, const armpl_int_t *ldz,
                    armpl_singlecomplex_t *work, const armpl_int_t *lwork,
                    float *rwork, const armpl_int_t *lrwork,
                    armpl_int_t *iwork, const armpl_int_t *liwork,
                    armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors. Not available in this release.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the Hermitian band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, AB is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the first superdiagonal and the diagonal of the tridiagonal matrix T are returned in rows KD and KD+1 of AB, and if UPLO = 'L', the diagonal and first subdiagonal of T are returned in the first two rows of AB.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD + 1.

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the orthonormal eigenvectors of the matrix A, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= 1, when N <= 1; otherwise If JOBZ = 'N' and N > 1, LWORK must be queried. LWORK = MAX(1, dimension) where dimension = (2KD+1)* N + KD*NTHREADS where KD is the size of the band. NTHREADS is the number of threads used when openMP compilation is enabled, otherwise =1. If JOBZ = 'V' and N > 1, LWORK must be queried. Not yet available.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array,. dimension (LRWORK) On exit, if INFO = 0, RWORK(1) returns the optimal LRWORK.

LRWORK Input parameter.

LRWORK is INTEGER

The dimension of array RWORK. If N <= 1, LRWORK must be at least 1. If JOBZ = 'N' and N > 1, LRWORK must be at least N. If JOBZ = 'V' and N > 1, LRWORK must be at least 1 + 5* N + 2*N**2.

If LRWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of array IWORK. If JOBZ = 'N' or N <= 1, LIWORK must be at least 1. If JOBZ = 'V' and N > 1, LIWORK must be at least 3 + 5* N .

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *zhbevd_2stage*. It also exists with a native C interface as *LAPACKE_chbevd_2stage*.

4.16.5 chbevx_2stage

CHBEVX_2STAGE computes selected eigenvalues and, optionally, eigenvectors of a complex Hermitian band matrix A using the 2stage technique for the reduction to tridiagonal. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine chbevx_2stage(JOBZ, RANGE, UPLO, N, KD, AB, LDAB, Q, LDQ, VL, VU,
                         IL, IU, ABSTOL, M, W, Z, LDZ, WORK, LWORK, RWORK,
                         IWORK, IFAIL, INFO)
```

C specification:

```
#include "armpl.h"

void chbevx_2stage_(const char *jobz, const char *range, const char *uplo,
                    const armpl_int_t *n, const armpl_int_t *kd,
                    armpl_singlecomplex_t *ab, const armpl_int_t *ldab,
                    armpl_singlecomplex_t *q, const armpl_int_t *ldq,
                    const float *vl, const float *vu, const armpl_int_t *il,
                    const armpl_int_t *iu, const float *abstol,
                    armpl_int_t *m, float *w, armpl_singlecomplex_t *z,
                    const armpl_int_t *ldz, armpl_singlecomplex_t *work,
                    const armpl_int_t *lwork, float *rwork,
                    armpl_int_t *iwork, armpl_int_t *ifail, armpl_int_t *info,
                    ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors. Not available in this release.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found; = 'V': all eigenvalues in the half-open interval (VL,VU] will be found; = 'I': the IL-th through IU-th eigenvalues will be found.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the Hermitian band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, AB is overwritten by values generated during the reduction to tridiagonal form.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD + 1.

Q Output parameter.

Q is COMPLEX

Q is an array, dimension (LDQ, N). If JOBZ = 'V', the N-by-N unitary matrix used in the reduction to tridiagonal form. If JOBZ = 'N', the array Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. If JOBZ = 'V', then LDQ >= max(1, N).

VL Input parameter.

VL is REAL

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is REAL

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is REAL

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing AB to tridiagonal form.

Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*SLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*SLAMCH('S').

See "Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy," by Demmel and Kahan, LAPACK Working Note #3.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is REAL

W is an array, dimension (N). The first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). If an eigenvector fails to converge, then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL. If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= 1, when N <= 1; otherwise If JOBZ = 'N' and N > 1, LWORK must be queried. LWORK = MAX(1, dimension) where dimension = (2KD+1)* N + KD*NTHREADS where KD is the size of the band. NTHREADS is the number of threads used when openMP compilation is enabled, otherwise =1. If JOBZ = 'V' and N > 1, LWORK must be queried. Not yet available.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (7*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (5*N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (N)

If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, then i eigenvectors failed to converge. Their indices are stored in array IFAIL.

Related Information

For this routine in other precisions, please see *zhbevx_2stage*. It also exists with a native C interface as *LAPACKE_chbevx_2stage*.

4.16.6 cheev_2stage

CHEEV_2STAGE computes all eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix A using the 2stage technique for the reduction to tridiagonal.

Syntax

Fortran specification:

```
use armpl_library

subroutine cheev_2stage(JOBZ, UPLO, N, A, LDA, W, WORK, LWORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cheev_2stage_(const char *jobz, const char *uplo, const armpl_int_t *n,
                   armpl_singlecomplex_t *a, const armpl_int_t *lda, float *w,
                   armpl_singlecomplex_t *work, const armpl_int_t *lwork,
                   float *rwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors. Not available in this release.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A. On exit, if JOBZ = 'V', then if INFO = 0, A contains the orthonormal eigenvectors of the matrix A. If JOBZ = 'N', then on exit the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= 1, when N <= 1; otherwise If JOBZ = 'N' and N > 1, LWORK must be queried. LWORK = MAX(1, dimension) where dimension = max(stage1,stage2) + (KD+1)* N + N = N*KD + N*max(KD+1,FACTOPTNB) + max(2*KD*KD, KD*NTHREADS) + (KD+1)* N + N where KD is the blocking size of the reduction, FACTOPTNB is the blocking used by the QR or LQ algorithm, usually FACTOPTNB=128 is a good choice NTHREADS is the number of threads used when openMP compilation is enabled, otherwise =1. If JOBZ = 'V' and N > 1, LWORK must be queried. Not yet available

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (max(1, 3*N-2)) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *zheev_2stage*. It also exists with a native C interface as *LAPACKE_cheev_2stage*.

4.16.7 cheevd_2stage

CHEEVD_2STAGE computes all eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix A using the 2stage technique for the reduction to tridiagonal. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine cheevd_2stage(JOBZ, UPLO, N, A, LDA, W, WORK, LWORK, RWORK, LRWORK,
                         IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cheevd_2stage_(const char *jobz, const char *uplo, const armpl_int_t *n,
                    armpl_singlecomplex_t *a, const armpl_int_t *lda,
                    float *w, armpl_singlecomplex_t *work,
                    const armpl_int_t *lwork, float *rwork,
                    const armpl_int_t *lrwork, armpl_int_t *iwork,
                    const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors. Not available in this release.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A. On exit, if JOBZ = 'V', then if INFO = 0, A contains the orthonormal eigenvectors of the matrix A. If JOBZ = 'N', then on exit the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If N <= 1, LWORK must be at least 1. If JOBZ = 'N' and N > 1, LWORK must be queried. LWORK = MAX(1, dimension) where dimension = max(stage1,stage2) + (KD+1)* N + N+1 = N*KD + N*max(KD+1,FACTOPTNB) + max(2*KD*KD, KD*NTHREADS) + (KD+1)* N + N+1 where KD is the blocking size of the reduction, FACTOPTNB is the blocking used by the QR or LQ algorithm, usually FACTOPTNB=128 is a good choice NTHREADS is the number of threads used when openMP compilation is enabled, otherwise =1. If JOBZ = 'V' and N > 1, LWORK must be at least 2* N + N**2

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array,. dimension (LRWORK) On exit, if INFO = 0, RWORK(1) returns the optimal LRWORK.

LRWORK Input parameter.

LRWORK is INTEGER

The dimension of the array RWORK. If N <= 1, LRWORK must be at least 1. If JOBZ = 'N' and N > 1, LRWORK must be at least N. If JOBZ = 'V' and N > 1, LRWORK must be at least 1 + 5* N + 2*N**2.

If LRWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. If N <= 1, LIWORK must be at least 1. If JOBZ = 'N' and N > 1, LIWORK must be at least 1. If JOBZ = 'V' and N > 1, LIWORK must be at least 3 + 5*N.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i and JOBZ = 'N', then the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero; if INFO = i and JOBZ = 'V', then the algorithm failed to compute an eigenvalue while working on the submatrix lying in rows and columns INFO/(N+1) through mod(INFO,N+1).

Related Information

For this routine in other precisions, please see *zheevd_2stage*. It also exists with a native C interface as *LAPACKE_cheevd_2stage*.

4.16.8 cheevr_2stage

CHEEVR_2STAGE computes selected eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix A using the 2stage technique for the reduction to tridiagonal. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

CHEEVR_2STAGE first reduces the matrix A to tridiagonal form T with a call to CHETRD. Then, whenever possible, CHEEVR_2STAGE calls CSTEMR to compute eigenspectrum using Relatively Robust Representations. CSTEMR computes eigenvalues by the dqds algorithm, while orthogonal eigenvectors are computed from various "good" L D L^T representations (also known as Relatively Robust Representations). Gram-Schmidt orthogonalization is avoided as far as possible. More specifically, the various steps of the algorithm are as follows.

For each unreduced block (submatrix) of T,

```
(a) Compute T - sigma I  = L D L^T, so that L and D
    define all the wanted eigenvalues to high relative accuracy.
    This means that small relative changes in the entries of D and L
    cause only small relative changes in the eigenvalues and
    eigenvectors. The standard (unfactored) representation of the
    tridiagonal matrix T does not have this property in general.
(b) Compute the eigenvalues to suitable accuracy.
    If the eigenvectors are desired, the algorithm attains full
    accuracy of the computed eigenvalues only right before
    the corresponding vectors have to be computed, see steps c) and d).
(c) For each cluster of close eigenvalues, select a new
    shift close to the cluster, find a new factorization, and refine
    the shifted eigenvalues to suitable accuracy.
(d) For each eigenvalue with a large enough relative separation compute
    the corresponding eigenvector by forming a rank revealing twisted
    factorization. Go back to (c) for any clusters that remain.
```

The desired accuracy of the output can be specified by the input parameter ABSTOL.

For more details, see DSTEMR's documentation and: - Inderjit S. Dhillon and Beresford N. Parlett: "Multiple representations

```
 to compute orthogonal eigenvectors of symmetric tridiagonal matrices,"
Linear Algebra and its Applications, 387(1), pp. 1-28, August 2004.
```

- Inderjit Dhillon and Beresford Parlett: "Orthogonal Eigenvectors and

```
 Relative Gaps," SIAM Journal on Matrix Analysis and Applications, Vol. 25,
2004.  Also LAPACK Working Note 154.
```

- Inderjit Dhillon: "A new O(n^2) algorithm for the symmetric

```
 tridiagonal eigenvalue/eigenvector problem",
Computer Science Division Technical Report No. UCB/CSD-97-971,
UC Berkeley, May 1997.
```

Note 1 : CHEEVR_2STAGE calls CSTEMR when the full spectrum is requested on machines which conform to the ieee-754 floating point standard. CHEEVR_2STAGE calls SSTEBZ and CSTEIN on non-ieee machines and when partial spectrum requests are made.

Normal execution of CSTEMR may create NaNs and infinities and hence may abort due to a floating point exception in environments which do not handle NaNs and infinities in the ieee standard default manner.

Syntax

Fortran specification:

```
use armpl_library

subroutine cheevr_2stage(JOBZ, RANGE, UPLO, N, A, LDA, VL, VU, IL, IU, ABSTOL,
                         M, W, Z, LDZ, ISUPPZ, WORK, LWORK, RWORK, LRWORK,
                         IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cheevr_2stage_(const char *jobz, const char *range, const char *uplo,
                    const armpl_int_t *n, armpl_singlecomplex_t *a,
                    const armpl_int_t *lda, const float *vl, const float *vu,
                    const armpl_int_t *il, const armpl_int_t *iu,
                    const float *abstol, armpl_int_t *m, float *w,
                    armpl_singlecomplex_t *z, const armpl_int_t *ldz,
                    armpl_int_t *isuppz, armpl_singlecomplex_t *work,
                    const armpl_int_t *lwork, float *rwork,
                    const armpl_int_t *lrwork, armpl_int_t *iwork,
                    const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors. Not available in this release.

RANGE Input parameter.

RANGE is CHARACTER*1

= ‘A’: all eigenvalues will be found. = ‘V’: all eigenvalues in the half-open interval (VL,VU] will be found. = ‘I’: the IL-th through IU-th eigenvalues will be found. For RANGE = ‘V’ or ‘I’ and IU - IL < N - 1, SSTEBZ and CSTEIN are called

UPLO Input parameter.

UPLO is CHARACTER*1

= ‘U’: Upper triangle of A is stored; = ‘L’: Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = ‘U’, the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = ‘L’, the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A. On exit, the lower triangle (if UPLO=’L’) or the upper triangle (if UPLO=’U’) of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

VL Input parameter.

VL is REAL

If RANGE=’V’, the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = ‘A’ or ‘I’.

VU Input parameter.

VU is REAL

If RANGE=’V’, the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = ‘A’ or ‘I’.

IL Input parameter.

IL is INTEGER

If RANGE=’I’, the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = ‘A’ or ‘V’.

IU Input parameter.

IU is INTEGER

If RANGE=’I’, the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = ‘A’ or ‘V’.

ABSTOL Input parameter.

ABSTOL is REAL

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing A to tridiagonal form.

See “Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy,” by Demmel and Kahan, LAPACK Working Note #3.

If high relative accuracy is important, set ABSTOL to SLAMCH('Safe minimum'). Doing so will guarantee that eigenvalues are computed to high relative accuracy when possible in future releases. The current code does not make any guarantees about high relative accuracy, but furutre releases will. See J. Barlow and J. Demmel, "Computing Accurate Eigensystems of Scaled Diagonally Dominant Matrices", LAPACK Working Note #7, for a discussion of which matrices define their eigenvalues to high relative accuracy.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is REAL

W is an array, dimension (N). The first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

ISUPPZ Output parameter.

ISUPPZ is INTEGER array, dimension (2*max(1, M))

The support of the eigenvectors in Z, i.e., the indices indicating the nonzero elements in Z. The i-th eigenvector is nonzero only in elements ISUPPZ(2*i-1) through ISUPPZ(2*i). This is an output of CSTEMR (tridiagonal matrix). The support of the eigenvectors of A is typically 1:N because of the unitary transformations applied by CUNMTR. Implemented only for RANGE = 'A' or 'I' and IU - IL = N - 1

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If JOBZ = 'N' and N > 1, LWORK must be queried. LWORK = MAX(1, 26*N, dimension) where dimension = max(stage1,stage2) + (KD+1)* N + N = N*KD + N*max(KD+1,FACTOPTNB) + max(2*KD*KD, KD*NTHREADS) + (KD+1)* N + N where KD is the blocking size of the reduction, FACTOPTNB is the blocking used by the QR or LQ algorithm, usually FACTOPTNB=128 is a good choice NTHREADS is the number of threads used when openMP compilation is enabled, otherwise =1. If JOBZ = 'V' and N > 1, LWORK must be queried. Not yet available

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (MAX(1, LRWORK)). On exit, if INFO = 0, RWORK(1) returns the optimal (and minimal) LRWORK.

LRWORK Input parameter.

LRWORK is INTEGER

The length of the array RWORK. LRWORK >= max(1,24*N).

If LRWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal (and minimal) LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. LIWORK >= max(1,10*N).

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: Internal error

Related Information

For this routine in other precisions, please see *zheevr_2stage*. It also exists with a native C interface as *LAPACKE_cheevr_2stage*.

4.16.9 cheevx_2stage

CHEEVX_2STAGE computes selected eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix A using the 2stage technique for the reduction to tridiagonal. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine cheevx_2stage(JOBZ, RANGE, UPLO, N, A, LDA, VL, VU, IL, IU, ABSTOL,
                         M, W, Z, LDZ, WORK, LWORK, RWORK, IWORK, IFAIL,
                         INFO)
```

C specification:

```
#include "armpl.h"

void cheevx_2stage_(const char *jobz, const char *range, const char *uplo,
                    const armpl_int_t *n, armpl_singlecomplex_t *a,
                    const armpl_int_t *lda, const float *vl, const float *vu,
```

(continues on next page)

(continued from previous page)

```
                    const armpl_int_t *il, const armpl_int_t *iu,
                    const float *abstol, armpl_int_t *m, float *w,
                    armpl_singlecomplex_t *z, const armpl_int_t *ldz,
                    armpl_singlecomplex_t *work, const armpl_int_t *lwork,
                    float *rwork, armpl_int_t *iwork, armpl_int_t *ifail,
                    armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors. Not available in this release.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found. = 'V': all eigenvalues in the half-open interval (VL,VU] will be found. = 'I': the IL-th through IU-th eigenvalues will be found.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A. On exit, the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

VL Input parameter.

VL is REAL

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is REAL

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is REAL

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing A to tridiagonal form.

Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*SLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*SLAMCH('S').

See "Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy," by Demmel and Kahan, LAPACK Working Note #3.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is REAL

W is an array, dimension (N). On normal exit, the first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). If an eigenvector fails to converge, then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL. If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= 1, when N <= 1; otherwise If JOBZ = 'N' and N > 1, LWORK must be queried. LWORK = MAX(1, 8*N, dimension) where dimension = max(stage1,stage2) + (KD+1)* N + N = N*KD + N*max(KD+1,FACTOPTNB) + max(2*KD*KD, KD*NTHREADS) + (KD+1)* N + N where KD

is the blocking size of the reduction, FACTOPTNB is the blocking used by the QR or LQ algorithm, usually FACTOPTNB=128 is a good choice NTHREADS is the number of threads used when openMP compilation is enabled, otherwise =1. If JOBZ = 'V' and N > 1, LWORK must be queried. Not yet available

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (7*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (5*N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (N)

If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, then i eigenvectors failed to converge. Their indices are stored in array IFAIL.

Related Information

For this routine in other precisions, please see *zheevx_2stage*. It also exists with a native C interface as *LAPACKE_cheevx_2stage*.

4.16.10 chegv_2stage

CHEGV_2STAGE computes all the eigenvalues, and optionally, the eigenvectors of a complex generalized Hermitian-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x. Here A and B are assumed to be Hermitian and B is also positive definite. This routine use the 2stage technique for the reduction to tridiagonal which showed higher performance on recent architecture and for large sizes N>2000.

Syntax

Fortran specification:

```
use armpl_library

subroutine chegv_2stage(ITYPE, JOBZ, UPLO, N, A, LDA, B, LDB, W, WORK, LWORK,
                        RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void chegv_2stage_(const armpl_int_t *itype, const char *jobz,
                   const char *uplo, const armpl_int_t *n,
                   armpl_singlecomplex_t *a, const armpl_int_t *lda,
                   armpl_singlecomplex_t *b, const armpl_int_t *ldb, float *w,
                   armpl_singlecomplex_t *work, const armpl_int_t *lwork,
                   float *rwork, armpl_int_t *info, ... );
```

Parameters

ITYPE Input parameter.

ITYPE is INTEGER

Specifies the problem type to be solved: = 1: A*x = (lambda)*B*x = 2: A*B*x = (lambda)*x = 3: B*A*x = (lambda)*x

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors. Not available in this release.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangles of A and B are stored; = 'L': Lower triangles of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A.

On exit, if JOBZ = 'V', then if INFO = 0, A contains the matrix Z of eigenvectors. The eigenvectors are normalized as follows: if ITYPE = 1 or 2, Z^H *B*Z = I; if ITYPE = 3, Z^H *inv(B)*Z = I. If JOBZ = 'N', then on exit the upper triangle (if UPLO='U') or the lower triangle (if UPLO='L') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, N). On entry, the Hermitian positive definite matrix B. If UPLO = 'U', the leading N-by-N upper triangular part of B contains the upper triangular part of the matrix B. If UPLO = 'L', the leading N-by-N lower triangular part of B contains the lower triangular part of the matrix B.

On exit, if INFO <= N, the part of B containing the matrix is overwritten by the triangular factor U or L from the Cholesky factorization B = U^H *U or B = L*L^H .

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= 1, when N <= 1; otherwise If JOBZ = 'N' and N > 1, LWORK must be queried. LWORK = MAX(1, dimension) where dimension = max(stage1,stage2) + (KD+1)* N + N = N*KD + N*max(KD+1,FACTOPTNB) + max(2*KD*KD, KD*NTHREADS) + (KD+1)* N + N where KD is the blocking size of the reduction, FACTOPTNB is the blocking used by the QR or LQ algorithm, usually FACTOPTNB=128 is a good choice NTHREADS is the number of threads used when openMP compilation is enabled, otherwise =1. If JOBZ = 'V' and N > 1, LWORK must be queried. Not yet available

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (max(1, 3*N-2)) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: CPOTRF or CHEEV returned an error code: <= N: if INFO = i, CHEEV failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero; > N: if INFO = N + i, for 1 <= i <= N, then the leading minor of order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *zhegv_2stage*. It also exists with a native C interface as *LAPACKE_chegv_2stage*.

4.16.11 chesv_aa_2stage

CHESV_AA_2STAGE computes the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N Hermitian matrix and X and B are N-by-NRHS matrices.

Aasen's 2-stage algorithm is used to factor A as

```
A = U * T * U**H,  if UPLO = 'U', or
A = L * T * L**H,  if UPLO = 'L',
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and T is Hermitian and band. The matrix T is then LU-factored with partial pivoting. The factored form of A is then used to solve the system of equations A * X = B.

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine chesv_aa_2stage(UPLO, N, NRHS, A, LDA, TB, LTB, IPIV, IPIV2, B,
                           LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void chesv_aa_2stage_(const char *uplo, const armpl_int_t *n,
                      const armpl_int_t *nrhs, armpl_singlecomplex_t *a,
                      const armpl_int_t *lda, armpl_singlecomplex_t *tb,
                      const armpl_int_t *ltb, armpl_int_t *ipiv,
                      armpl_int_t *ipiv2, armpl_singlecomplex_t *b,
                      const armpl_int_t *ldb, armpl_singlecomplex_t *work,
                      const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, L is stored below (or above) the subdiaonal blocks, when UPLO is 'L' (or 'U').

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TB Output parameter.

TB is COMPLEX

TB is an array, dimension (LTB). On exit, details of the LU factorization of the band matrix.

LTB Input parameter.

The size of the array TB. LTB >= 4*N, internally

used to select NB such that LTB >= (3*NB+1)*N.

If LTB = -1, then a workspace query is assumed; the routine only calculates the optimal size of LTB, returns this value as the first entry of TB, and no error message related to LTB is issued by XERBLA.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

On exit, it contains the details of the interchanges, i.e., the row and column k of A were interchanged with the row and column IPIV(k).

IPIV2 Output parameter.

IPIV is INTEGER array, dimension (N)

On exit, it contains the details of the interchanges, i.e., the row and column k of T were interchanged with the row and column IPIV(k).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Output parameter.

WORK is COMPLEX workspace of size LWORK

LWORK Input parameter.

The size of WORK. LWORK >= N, internally used to select NB

such that LWORK >= N*NB.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, band LU factorization failed on i-th column

Related Information

For this routine in other precisions, please see *zhesv_aa_2stage*. It also exists with a native C interface as *LAPACKE_chesv_aa_2stage*.

4.16.12 chetf2_rk

CHETF2_RK computes the factorization of a complex Hermitian matrix A using the bounded Bunch-Kaufman (rook) diagonal pivoting method:

```
A = P*U*D*(U**H)*(P**T) or A = P*L*D*(L**H)*(P**T),
```

where U (or L) is unit upper (or lower) triangular matrix, U^H (or L^H) is the conjugate of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is Hermitian and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the unblocked version of the algorithm, calling Level 2 BLAS. For more information see Further Details section.

Syntax

Fortran specification:

```
use armpl_library

subroutine chetf2_rk(UPLO, N, A, LDA, E, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void chetf2_rk_(const char *uplo, const armpl_int_t *n,
                armpl_singlecomplex_t *a, const armpl_int_t *lda,
                armpl_singlecomplex_t *e, armpl_int_t *ipiv,
                armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the Hermitian matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U': the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced.

If UPLO = 'L': the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, contains: a) ONLY diagonal elements of the Hermitian block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Output parameter.

E is COMPLEX

E is an array, dimension (N). On exit, contains the superdiagonal (or subdiagonal) elements of the Hermitian block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) is set to 0; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) is set to 0.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is set to 0 in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

IPIV describes the permutation matrix P in the factorization of matrix A as follows. The absolute value of IPIV(k) represents the index of row and column that were interchanged with the k-th row and column. The value of UPLO describes the order in which the interchanges were applied. Also, the sign of IPIV represents the block structure of the Hermitian block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks which correspond to 1 or 2 interchanges at each factorization step. For more info see Further Details section.

If UPLO = 'U', (in factorization order, k decreases from N to 1): a) A single positive entry IPIV(k) > 0 means: D(k,k) is a 1-by-1 diagonal block. If IPIV(k) != k, rows and columns k and IPIV(k) were interchanged in the matrix A(1:N,1:N); If IPIV(k) = k, no interchange occurred.

b) A pair of consecutive negative entries IPIV(k) < 0 and IPIV(k-1) < 0 means: D(k-1:k,k-1:k) is a 2-by-2 diagonal block. (NOTE: negative entries in IPIV appear ONLY in pairs). 1) If -IPIV(k) != k, rows and columns k and -IPIV(k) were interchanged in the matrix A(1:N,1:N). If -IPIV(k) = k, no interchange occurred. 2) If -IPIV(k-1) != k-1, rows and columns k-1 and -IPIV(k-1) were interchanged in the matrix A(1:N,1:N). If -IPIV(k-1) = k-1, no interchange occurred.

c) In both cases a) and b), always ABS(IPIV(k)) <= k.

d) NOTE: Any entry IPIV(k) is always NONZERO on output.

If UPLO = 'L', (in factorization order, k increases from 1 to N): a) A single positive entry IPIV(k) > 0 means: D(k,k) is a 1-by-1 diagonal block. If IPIV(k) != k, rows and columns k and IPIV(k) were interchanged in the matrix A(1:N,1:N). If IPIV(k) = k, no interchange occurred.

b) A pair of consecutive negative entries IPIV(k) < 0 and IPIV(k+1) < 0 means: D(k:k+1,k:k+1) is a 2-by-2 diagonal block. (NOTE: negative entries in IPIV appear ONLY in pairs). 1) If -IPIV(k) != k, rows and columns k and -IPIV(k) were interchanged in the matrix A(1:N,1:N). If -IPIV(k) = k, no interchange occurred. 2) If -IPIV(k+1) != k+1, rows and columns k-1 and -IPIV(k-1) were interchanged in the matrix A(1:N,1:N). If -IPIV(k+1) = k+1, no interchange occurred.

c) In both cases a) and b), always ABS(IPIV(k)) >= k.

d) NOTE: Any entry IPIV(k) is always NONZERO on output.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit

< 0: If INFO = -k, the k-th argument had an illegal value

> 0: If INFO = k, the matrix A is singular, because: If UPLO = 'U': column k in the upper triangular part of A contains all zeros. If UPLO = 'L': column k in the lower triangular part of A contains all zeros.

Therefore D(k,k) is exactly zero, and superdiagonal elements of column k of U (or subdiagonal elements of column k of L) are all zeros. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

NOTE: INFO only stores the first occurrence of a singularity, any subsequent occurrence of singularity is not stored in INFO even though the factorization always completes.

Related Information

For this routine in other precisions, please see *zhetf2_rk*.

4.16.13 chetrd_2stage

CHETRD_2STAGE reduces a complex Hermitian matrix A to real symmetric tridiagonal form T by a unitary similarity transformation: $Q1^H$ $Q2^H$ * A * Q2 * Q1 = T.

Syntax

Fortran specification:

```
use armpl_library

subroutine chetrd_2stage(VECT, UPLO, N, A, LDA, D, E, TAU, HOUS2, LHOUS2,
                         WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void chetrd_2stage_(const char *vect, const char *uplo, const armpl_int_t *n,
                    armpl_singlecomplex_t *a, const armpl_int_t *lda,
                    float *d, float *e, armpl_singlecomplex_t *tau,
                    armpl_singlecomplex_t *hous2, const armpl_int_t *lhous2,
                    armpl_singlecomplex_t *work, const armpl_int_t *lwork,
                    armpl_int_t *info, ... );
```

Parameters

VECT Input parameter.

VECT is CHARACTER*1

= 'N': No need for the Housholder representation, in particular for the second stage (Band to tridiagonal) and thus LHOUS2 is of size max(1, 4*N); = 'V': the Householder representation is needed to either generate Q1 Q2 or to apply Q1 Q2, then LHOUS2 is to be queried and computed. (NOT AVAILABLE IN THIS RELEASE).

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, if UPLO = 'U', the band superdiagonal of A are overwritten by the corresponding elements of the internal band-diagonal matrix AB, and the elements above the KD superdiagonal, with the array TAU, represent the unitary matrix Q1 as a product of elementary reflectors; if UPLO = 'L', the diagonal and band subdiagonal of A are overwritten by the corresponding elements of the internal band-diagonal matrix AB, and the elements below the KD subdiagonal, with the array TAU, represent the unitary matrix Q1 as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

D Output parameter.

D is REAL

D is an array, dimension (N). The diagonal elements of the tridiagonal matrix T.

E Output parameter.

E is REAL

E is an array, dimension (N-1). The off-diagonal elements of the tridiagonal matrix T.

TAU Output parameter.

TAU is COMPLEX

TAU is an array, dimension (N-KD). The scalar factors of the elementary reflectors of the first stage (see Further Details).

HOUS2 Output parameter.

HOUS2 is COMPLEX

HOUS2 is an array, dimension LHOUS2, that. store the Householder representation of the stage2 band to tridiagonal.

LHOUS2 Input parameter.

LHOUS2 is INTEGER

The dimension of the array HOUS2. LHOUS2 = MAX(1, dimension) If LWORK = -1, or LHOUS2=-1, then a query is assumed; the routine only calculates the optimal size of the HOUS2 array, returns this value as the first entry of the HOUS2 array, and no error message related to LHOUS2 is issued by XERBLA. LHOUS2 = MAX(1, dimension) where dimension = 4* N if VECT='N' not available now if VECT='H'

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK = MAX(1, dimension) If LWORK = -1, or LHOUS2=-1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA. LWORK = MAX(1, dimension) where dimension = max(stage1,stage2) + (KD+1)*N = N*KD + N*max(KD+1,FACTOPTNB) + max(2*KD*KD, KD*NTHREADS) + (KD+1)* N where KD is the blocking size of the reduction, FACTOPTNB is the blocking used by the QR or LQ algorithm, usually FACTOPTNB=128 is a good choice NTHREADS is the number of threads used when openMP compilation is enabled, otherwise =1.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zhetrd_2stage*.

4.16.14 chetrd_he2hb

CHETRD_HE2HB reduces a complex Hermitian matrix A to complex Hermitian band-diagonal form AB by a unitary similarity transformation: Q^H * A * Q = AB.

Syntax

Fortran specification:

```
use armpl_library

subroutine chetrd_he2hb(UPLO, N, KD, A, LDA, AB, LDAB, TAU, WORK, LWORK,
                        INFO)
```

C specification:

```
#include "armpl.h"

void chetrd_he2hb_(const char *uplo, const armpl_int_t *n,
                   const armpl_int_t *kd, armpl_singlecomplex_t *a,
                   const armpl_int_t *lda, armpl_singlecomplex_t *ab,
                   const armpl_int_t *ldab, armpl_singlecomplex_t *tau,
                   armpl_singlecomplex_t *work, const armpl_int_t *lwork,
                   armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the reduced matrix if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0. The reduced matrix is stored in the array AB.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, if UPLO = 'U', the diagonal and first superdiagonal of A are overwritten by the corresponding elements of the tridiagonal matrix T, and the elements above the first superdiagonal, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors; if UPLO = 'L', the diagonal and first subdiagonal of A are over- written by the corresponding elements of the tridiagonal matrix T, and the elements below the first subdiagonal, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AB Output parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). On exit, the upper or lower triangle of the Hermitian band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB

as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

TAU Output parameter.

TAU is COMPLEX

TAU is an array, dimension (N-KD). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (LWORK). On exit, if INFO = 0, or if LWORK=-1, WORK(1) returns the size of LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK which should be calculated by a workspace query. LWORK = MAX(1, LWORK_QUERY) If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA. LWORK_QUERY = N* KD + N*max(KD,FACTOPTNB) + 2*KD*KD where FACTOPTNB is the blocking used by the QR or LQ algorithm, usually FACTOPTNB=128 is a good choice otherwise putting LWORK=-1 will provide the size of WORK.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zhetrd_he2hb*.

4.16.15 chetrf_aa_2stage

CHETRF_AA_2STAGE computes the factorization of a real hermitian matrix A using the Aasen's algorithm. The form of the factorization is

```
A = U*T*U**T  or  A = L*T*L**T
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and T is a hermitian band matrix with the bandwidth of NB (NB is internally selected and stored in TB(1), and T is LU factorized with partial pivoting).

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine chetrf_aa_2stage(UPLO, N, A, LDA, TB, LTB, IPIV, IPIV2, WORK,
                            LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void chetrf_aa_2stage_(const char *uplo, const armpl_int_t *n,
                       armpl_singlecomplex_t *a, const armpl_int_t *lda,
                       armpl_singlecomplex_t *tb, const armpl_int_t *ltb,
                       armpl_int_t *ipiv, armpl_int_t *ipiv2,
                       armpl_singlecomplex_t *work, const armpl_int_t *lwork,
                       armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, L is stored below (or above) the subdiaonal blocks, when UPLO is 'L' (or 'U').

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TB Output parameter.

TB is COMPLEX

TB is an array, dimension (LTB). On exit, details of the LU factorization of the band matrix.

LTB Input parameter.

The size of the array TB. LTB >= 4*N, internally

used to select NB such that LTB >= (3*NB+1)*N.

If LTB = -1, then a workspace query is assumed; the routine only calculates the optimal size of LTB, returns this value as the first entry of TB, and no error message related to LTB is issued by XERBLA.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

On exit, it contains the details of the interchanges, i.e., the row and column k of A were interchanged with the row and column IPIV(k).

IPIV2 Output parameter.

IPIV is INTEGER array, dimension (N)

On exit, it contains the details of the interchanges, i.e., the row and column k of T were interchanged with the row and column IPIV(k).

WORK Output parameter.

WORK is COMPLEX workspace of size LWORK

LWORK Input parameter.

The size of WORK. LWORK >= N, internally used to select NB

such that LWORK >= N*NB.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, band LU factorization failed on i-th column

Related Information

For this routine in other precisions, please see *zhetrf_aa_2stage*. It also exists with a native C interface as *LAPACKE_chetrf_aa_2stage*.

4.16.16 chetri_3x

CHETRI_3X computes the inverse of a complex Hermitian indefinite matrix A using the factorization computed by CHETRF_RK or CHETRF_BK:

```
A = P*U*D*(U**H)*(P**T) or A = P*L*D*(L**H)*(P**T),
```

where U (or L) is unit upper (or lower) triangular matrix, U^H (or L^H) is the conjugate of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is Hermitian and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine chetri_3x(UPLO, N, A, LDA, E, IPIV, WORK, NB, INFO)
```

C specification:

```
#include "armpl.h"

void chetri_3x_(const char *uplo, const armpl_int_t *n,
                armpl_singlecomplex_t *a, const armpl_int_t *lda,
                const armpl_singlecomplex_t *e, const armpl_int_t *ipiv,
                armpl_singlecomplex_t *work, const armpl_int_t *nb,
                armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, diagonal of the block diagonal matrix D and factors U or L as computed by CHETRF_RK and CHETRF_BK: a) ONLY diagonal elements of the Hermitian block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D should be provided on entry in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

On exit, if INFO = 0, the Hermitian inverse of the original matrix. If UPLO = 'U': the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; If UPLO = 'L': the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Input parameter.

E is COMPLEX

E is an array, dimension (N). On entry, contains the superdiagonal (or subdiagonal) elements of the Hermitian block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) not referenced; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) not referenced.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is not referenced in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CHETRF_RK or CHETRF_BK.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (N+NB+1,NB+3). .

NB Input parameter.

NB is INTEGER

Block size.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *zhetri_3x*.

4.16.17 chetrs_aa_2stage

CHETRS_AA_2STAGE solves a system of linear equations A*X = B with a real hermitian matrix A using the factorization A = U*T*U^T or A = L*T*L^T computed by CHETRF_AA_2STAGE.

Syntax

Fortran specification:

```
use armpl_library

subroutine chetrs_aa_2stage(UPLO, N, NRHS, A, LDA, TB, LTB, IPIV, IPIV2, B,
                            LDB, INFO)
```

C specification:

```
#include "armpl.h"

void chetrs_aa_2stage_(const char *uplo, const armpl_int_t *n,
                       const armpl_int_t *nrhs,
                       const armpl_singlecomplex_t *a, const armpl_int_t *lda,
                       armpl_singlecomplex_t *tb, const armpl_int_t *ltb,
                       const armpl_int_t *ipiv, const armpl_int_t *ipiv2,
                       armpl_singlecomplex_t *b, const armpl_int_t *ldb,
                       armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*T*U^T ; = 'L': Lower triangular, form is A = L*T*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). Details of factors computed by CHETRF_AA_2STAGE.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TB Output parameter.

TB is COMPLEX

TB is an array, dimension (LTB). Details of factors computed by CHETRF_AA_2STAGE.

LTB Input parameter.

The size of the array TB. LTB >= 4*N.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges as computed by CHETRF_AA_2STAGE.

IPIV2 Input parameter.

IPIV2 is INTEGER array, dimension (N)

Details of the interchanges as computed by CHETRF_AA_2STAGE.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zhetrs_aa_2stage*. It also exists with a native C interface as *LAPACKE_chetrs_aa_2stage*.

4.16.18 chptrs

chptrs solves a system of linear equations A*X = B with a complex Hermitian matrix A stored in packed format using the factorization A = U*D*U^H or A = L*D*L^H computed by CHPTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine chptrs(UPLO, N, NRHS, AP, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void chptrs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_singlecomplex_t *ap, const armpl_int_t *ipiv,
             armpl_singlecomplex_t *b, const armpl_int_t *ldb,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^H ; = 'L': Lower triangular, form is A = L*D*L^H .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AP Input parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CHPTRF, stored as a packed triangular matrix.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CHPTRF.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zhptrs*. It also exists with a native C interface as *LAPACKE_chptrs*.

4.16.19 clahef_rk

CLAHEF_RK computes a partial factorization of a complex Hermitian matrix A using the bounded Bunch-Kaufman (rook) diagonal pivoting method. The partial factorization has the form:

A = (I U12) (A11 0) (I 0) if UPLO = 'U', or:

```
( 0  U22 ) (  0   D  ) ( U12**H U22**H )
```

A = (L11 0) (D 0) ($L11^H$ $L21^H$) if UPLO = 'L',

```
( L21  I ) (  0  A22 ) (  0       I    )
```

where the order of D is at most NB. The actual order is returned in the argument KB, and is either NB or NB-1, or N if N <= NB.

CLAHEF_RK is an auxiliary routine called by CHETRF_RK. It uses blocked code (calling Level 3 BLAS) to update the submatrix A11 (if UPLO = 'U') or A22 (if UPLO = 'L').

Syntax

Fortran specification:

```
use armpl_library

subroutine clahef_rk(UPLO, N, NB, KB, A, LDA, E, IPIV, W, LDW, INFO)
```

C specification:

```
#include "armpl.h"

void clahef_rk_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nb,
                armpl_int_t *kb, armpl_singlecomplex_t *a,
                const armpl_int_t *lda, armpl_singlecomplex_t *e,
                armpl_int_t *ipiv, armpl_singlecomplex_t *w,
                const armpl_int_t *ldw, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the Hermitian matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NB Input parameter.

NB is INTEGER

The maximum number of columns of the matrix A that should be factored. NB should be at least 2 to allow for 2-by-2 pivot blocks.

KB Output parameter.

KB is INTEGER

The number of columns of A that were actually factored. KB is either NB-1 or NB, or N if N <= NB.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U': the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced.

If UPLO = 'L': the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, contains: a) ONLY diagonal elements of the Hermitian block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Output parameter.

E is COMPLEX

E is an array, dimension (N). On exit, contains the superdiagonal (or subdiagonal) elements of the Hermitian block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) is set to 0; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) is set to 0.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is set to 0 in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

IPIV describes the permutation matrix P in the factorization of matrix A as follows. The absolute value of IPIV(k) represents the index of row and column that were interchanged with the k-th row and column. The value of UPLO describes the order in which the interchanges were applied. Also, the sign of IPIV represents the block structure of the Hermitian block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks which correspond to 1 or 2 interchanges at each factorization step.

If UPLO = 'U', (in factorization order, k decreases from N to 1): a) A single positive entry IPIV(k) > 0 means: D(k,k) is a 1-by-1 diagonal block. If IPIV(k) != k, rows and columns k and IPIV(k) were interchanged in the submatrix A(1:N,N-KB+1:N); If IPIV(k) = k, no interchange occurred.

b) A pair of consecutive negative entries IPIV(k) < 0 and IPIV(k-1) < 0 means: D(k-1:k,k-1:k) is a 2-by-2 diagonal block. (NOTE: negative entries in IPIV appear ONLY in pairs). 1) If -IPIV(k) != k, rows and columns k and -IPIV(k) were interchanged in the matrix A(1:N,N-KB+1:N). If -IPIV(k) = k, no interchange occurred. 2) If -IPIV(k-1) != k-1, rows and columns k-1 and -IPIV(k-1) were interchanged in the submatrix A(1:N,N-KB+1:N). If -IPIV(k-1) = k-1, no interchange occurred.

c) In both cases a) and b) is always ABS(IPIV(k)) <= k.

d) NOTE: Any entry IPIV(k) is always NONZERO on output.

If UPLO = 'L', (in factorization order, k increases from 1 to N): a) A single positive entry IPIV(k) > 0 means: D(k,k) is a 1-by-1 diagonal block. If IPIV(k) != k, rows and columns k and IPIV(k) were interchanged in the submatrix A(1:N,1:KB). If IPIV(k) = k, no interchange occurred.

b) A pair of consecutive negative entries IPIV(k) < 0 and IPIV(k+1) < 0 means: D(k:k+1,k:k+1) is a 2-by-2 diagonal block. (NOTE: negative entries in IPIV appear ONLY in pairs). 1) If -IPIV(k) != k, rows and columns k and -IPIV(k) were interchanged in the submatrix A(1:N,1:KB). If -IPIV(k) = k, no interchange occurred. 2) If -IPIV(k+1) != k+1, rows and columns k-1 and -IPIV(k-1) were interchanged in the submatrix A(1:N,1:KB). If -IPIV(k+1) = k+1, no interchange occurred.

c) In both cases a) and b) is always ABS(IPIV(k)) >= k.

d) NOTE: Any entry IPIV(k) is always NONZERO on output.

W Output parameter.

W is COMPLEX

W is an array, dimension (LDW, NB) .

LDW Input parameter.

LDW is INTEGER

The leading dimension of the array W. LDW >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit

< 0: If INFO = -k, the k-th argument had an illegal value

> 0: If INFO = k, the matrix A is singular, because: If UPLO = 'U': column k in the upper triangular part of A contains all zeros. If UPLO = 'L': column k in the lower triangular part of A contains all zeros.

Therefore D(k,k) is exactly zero, and superdiagonal elements of column k of U (or subdiagonal elements of column k of L) are all zeros. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

NOTE: INFO only stores the first occurrence of a singularity, any subsequent occurrence of singularity is not stored in INFO even though the factorization always completes.

Related Information

For this routine in other precisions, please see *zlahef_rk*.

4.16.20 clasyf_rk

CLASYF_RK computes a partial factorization of a complex symmetric matrix A using the bounded Bunch-Kaufman (rook) diagonal pivoting method. The partial factorization has the form:

A = (I U12) (A11 0) (I 0) if UPLO = 'U', or:

```
( 0  U22 ) (  0   D  ) ( U12**T U22**T )
```

A = (L11 0) (D 0) ($L11^T$ $L21^T$) if UPLO = 'L',

```
( L21  I ) (  0  A22 ) (  0       I    )
```

where the order of D is at most NB. The actual order is returned in the argument KB, and is either NB or NB-1, or N if N <= NB.

CLASYF_RK is an auxiliary routine called by CSYTRF_RK. It uses blocked code (calling Level 3 BLAS) to update the submatrix A11 (if UPLO = 'U') or A22 (if UPLO = 'L').

Syntax

Fortran specification:

```
use armpl_library

subroutine clasyf_rk(UPLO, N, NB, KB, A, LDA, E, IPIV, W, LDW, INFO)
```

C specification:

```
#include "armpl.h"

void clasyf_rk_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nb,
                armpl_int_t *kb, armpl_singlecomplex_t *a,
                const armpl_int_t *lda, armpl_singlecomplex_t *e,
                armpl_int_t *ipiv, armpl_singlecomplex_t *w,
                const armpl_int_t *ldw, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NB Input parameter.

NB is INTEGER

The maximum number of columns of the matrix A that should be factored. NB should be at least 2 to allow for 2-by-2 pivot blocks.

KB Output parameter.

KB is INTEGER

The number of columns of A that were actually factored. KB is either NB-1 or NB, or N if N <= NB.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U': the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced.

If UPLO = 'L': the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, contains: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Output parameter.

E is COMPLEX

E is an array, dimension (N). On exit, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) is set to 0; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) is set to 0.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is set to 0 in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

IPIV describes the permutation matrix P in the factorization of matrix A as follows. The absolute value of IPIV(k) represents the index of row and column that were interchanged with the k-th row and column. The value of UPLO describes the order in which the interchanges were applied. Also, the sign of IPIV represents the block structure of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks which correspond to 1 or 2 interchanges at each factorization step.

If UPLO = 'U', (in factorization order, k decreases from N to 1): a) A single positive entry IPIV(k) > 0 means: D(k,k) is a 1-by-1 diagonal block. If IPIV(k) != k, rows and columns k and IPIV(k) were interchanged in the submatrix A(1:N,N-KB+1:N); If IPIV(k) = k, no interchange occurred.

b) A pair of consecutive negative entries IPIV(k) < 0 and IPIV(k-1) < 0 means: D(k-1:k,k-1:k) is a 2-by-2 diagonal block. (NOTE: negative entries in IPIV appear ONLY in pairs). 1) If -IPIV(k) != k, rows and columns k and -IPIV(k) were interchanged in the matrix A(1:N,N-KB+1:N). If -IPIV(k) = k, no interchange

occurred. 2) If -IPIV(k-1) != k-1, rows and columns k-1 and -IPIV(k-1) were interchanged in the submatrix A(1:N,N-KB+1:N). If -IPIV(k-1) = k-1, no interchange occurred.

c) In both cases a) and b) is always ABS(IPIV(k)) <= k.

d) NOTE: Any entry IPIV(k) is always NONZERO on output.

If UPLO = 'L', (in factorization order, k increases from 1 to N): a) A single positive entry IPIV(k) > 0 means: D(k,k) is a 1-by-1 diagonal block. If IPIV(k) != k, rows and columns k and IPIV(k) were interchanged in the submatrix A(1:N,1:KB). If IPIV(k) = k, no interchange occurred.

b) A pair of consecutive negative entries IPIV(k) < 0 and IPIV(k+1) < 0 means: D(k:k+1,k:k+1) is a 2-by-2 diagonal block. (NOTE: negative entries in IPIV appear ONLY in pairs). 1) If -IPIV(k) != k, rows and columns k and -IPIV(k) were interchanged in the submatrix A(1:N,1:KB). If -IPIV(k) = k, no interchange occurred. 2) If -IPIV(k+1) != k+1, rows and columns k-1 and -IPIV(k-1) were interchanged in the submatrix A(1:N,1:KB). If -IPIV(k+1) = k+1, no interchange occurred.

c) In both cases a) and b) is always ABS(IPIV(k)) >= k.

d) NOTE: Any entry IPIV(k) is always NONZERO on output.

W Output parameter.

W is COMPLEX

W is an array, dimension (LDW, NB) .

LDW Input parameter.

LDW is INTEGER

The leading dimension of the array W. LDW >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit

< 0: If INFO = -k, the k-th argument had an illegal value

> 0: If INFO = k, the matrix A is singular, because: If UPLO = 'U': column k in the upper triangular part of A contains all zeros. If UPLO = 'L': column k in the lower triangular part of A contains all zeros.

Therefore D(k,k) is exactly zero, and superdiagonal elements of column k of U (or subdiagonal elements of column k of L) are all zeros. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

NOTE: INFO only stores the first occurrence of a singularity, any subsequent occurrence of singularity is not stored in INFO even though the factorization always completes.

Related Information

For this routine in other precisions, please see *dlasyf_rk*, *slasyf_rk* and *zlasyf_rk*.

4.16.21 csyconvf

If parameter WAY = 'C': CSYCONVF converts the factorization output format used in CSYTRF provided on entry in parameter A into the factorization output format used in CSYTRF_RK (or CSYTRF_BK) that is stored on exit in parameters A and E. It also coverts in place details of the intechanges stored in IPIV from the format used in CSYTRF into the format used in CSYTRF_RK (or CSYTRF_BK).

If parameter WAY = 'R': CSYCONVF performs the conversion in reverse direction, i.e. converts the factorization output format used in CSYTRF_RK (or CSYTRF_BK) provided on entry in parameters A and E into the factorization output format used in CSYTRF that is stored on exit in parameter A. It also coverts in place details of the intechanges stored in IPIV from the format used in CSYTRF_RK (or CSYTRF_BK) into the format used in CSYTRF.

CSYCONVF can also convert in Hermitian matrix case, i.e. between formats used in CHETRF and CHETRF_RK (or CHETRF_BK).

Syntax

Fortran specification:

```
use armpl_library

subroutine csyconvf(UPLO, WAY, N, A, LDA, E, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void csyconvf_(const char *uplo, const char *way, const armpl_int_t *n,
               armpl_singlecomplex_t *a, const armpl_int_t *lda,
               armpl_singlecomplex_t *e, armpl_int_t *ipiv, armpl_int_t *info,
               ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix A. = 'U': Upper triangular = 'L': Lower triangular

WAY Input parameter.

WAY is CHARACTER*1

= 'C': Convert = 'R': Revert

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). 1) If WAY ='C':

On entry, contains factorization details in format used in CSYTRF: a) all elements of the symmetric block diagonal matrix D on the diagonal of A and on superdiagonal (or subdiagonal) of A, and b) If UPLO = 'U': multipliers used to obtain factor U in the superdiagonal part of A. If UPLO = 'L': multipliers used to obtain factor L in the superdiagonal part of A.

On exit, contains factorization details in format used in CSYTRF_RK or CSYTRF_BK: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

2) If WAY = 'R':

On entry, contains factorization details in format used in CSYTRF_RK or CSYTRF_BK: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

On exit, contains factorization details in format used in CSYTRF: a) all elements of the symmetric block diagonal matrix D on the diagonal of A and on superdiagonal (or subdiagonal) of A, and b) If UPLO = 'U': multipliers used to obtain factor U in the superdiagonal part of A. If UPLO = 'L': multipliers used to obtain factor L in the superdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Input and output parameter.

E is COMPLEX

E is an array, dimension (N). 1) If WAY ='C':

On entry, just a workspace.

On exit, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) is set to 0; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) is set to 0.

2) If WAY = 'R':

On entry, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i),i=2:N, E(1) not referenced; If UPLO = 'L': E(i) = D(i+1,i),i=1:N-1, E(N) not referenced.

On exit, is not changed

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

1) If WAY ='C': On entry, details of the interchanges and the block structure of D in the format used in CSYTRF. On exit, details of the interchanges and the block structure of D in the format used in CSYTRF_RK (or CSYTRF_BK).

1) If WAY ='R': On entry, details of the interchanges and the block structure of D in the format used in CSYTRF_RK (or CSYTRF_BK). On exit, details of the interchanges and the block structure of D in the format used in CSYTRF.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dsyconvf*, *ssyconvf* and *zsyconvf*.

4.16.22 csyconvf_rook

If parameter WAY = 'C': CSYCONVF_ROOK converts the factorization output format used in CSYTRF_ROOK provided on entry in parameter A into the factorization output format used in CSYTRF_RK (or CSYTRF_BK) that is stored on exit in parameters A and E. IPIV format for CSYTRF_ROOK and CSYTRF_RK (or CSYTRF_BK) is the same and is not converted.

If parameter WAY = 'R': CSYCONVF_ROOK performs the conversion in reverse direction, i.e. converts the factorization output format used in CSYTRF_RK (or CSYTRF_BK) provided on entry in parameters A and E into the factorization output format used in CSYTRF_ROOK that is stored on exit in parameter A. IPIV format for CSYTRF_ROOK and CSYTRF_RK (or CSYTRF_BK) is the same and is not converted.

CSYCONVF_ROOK can also convert in Hermitian matrix case, i.e. between formats used in CHETRF_ROOK and CHETRF_RK (or CHETRF_BK).

Syntax

Fortran specification:

```
use armpl_library

subroutine csyconvf_rook(UPLO, WAY, N, A, LDA, E, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void csyconvf_rook_(const char *uplo, const char *way, const armpl_int_t *n,
                    armpl_singlecomplex_t *a, const armpl_int_t *lda,
                    armpl_singlecomplex_t *e, const armpl_int_t *ipiv,
                    armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix A. = 'U': Upper triangular = 'L': Lower triangular

WAY Input parameter.

WAY is CHARACTER*1

= 'C': Convert = 'R': Revert

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). 1) If WAY ='C':

On entry, contains factorization details in format used in CSYTRF_ROOK: a) all elements of the symmetric block diagonal matrix D on the diagonal of A and on superdiagonal (or subdiagonal) of A, and b) If UPLO = 'U': multipliers used to obtain factor U in the superdiagonal part of A. If UPLO = 'L': multipliers used to obtain factor L in the superdiagonal part of A.

On exit, contains factorization details in format used in CSYTRF_RK or CSYTRF_BK: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

2) If WAY = 'R':

On entry, contains factorization details in format used in CSYTRF_RK or CSYTRF_BK: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

On exit, contains factorization details in format used in CSYTRF_ROOK: a) all elements of the symmetric block diagonal matrix D on the diagonal of A and on superdiagonal (or subdiagonal) of A, and b) If UPLO = 'U': multipliers used to obtain factor U in the superdiagonal part of A. If UPLO = 'L': multipliers used to obtain factor L in the superdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Input and output parameter.

E is COMPLEX

E is an array, dimension (N). 1) If WAY ='C':

On entry, just a workspace.

On exit, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) is set to 0; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) is set to 0.

2) If WAY = 'R':

On entry, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i),i=2:N, E(1) not referenced; If UPLO = 'L': E(i) = D(i+1,i),i=1:N-1, E(N) not referenced.

On exit, is not changed

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

On entry, details of the interchanges and the block structure of D as determined: 1) by CSYTRF_ROOK, if WAY ='C'; 2) by CSYTRF_RK (or CSYTRF_BK), if WAY ='R'. The IPIV format is the same for all these routines.

On exit, is not changed.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dsyconvf_rook*, *ssyconvf_rook* and *zsyconvf_rook*.

4.16.23 csysv_aa_2stage

CSYSV_AA_2STAGE computes the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N symmetric matrix and X and B are N-by-NRHS matrices.

Aasen's 2-stage algorithm is used to factor A as

```
A = U * T * U**H,  if UPLO = 'U', or
A = L * T * L**H,  if UPLO = 'L',
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and T is symmetric and band. The matrix T is then LU-factored with partial pivoting. The factored form of A is then used to solve the system of equations A * X = B.

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine csysv_aa_2stage(UPLO, N, NRHS, A, LDA, TB, LTB, IPIV, IPIV2, B,
                           LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void csysv_aa_2stage_(const char *uplo, const armpl_int_t *n,
                      const armpl_int_t *nrhs, armpl_singlecomplex_t *a,
                      const armpl_int_t *lda, armpl_singlecomplex_t *tb,
                      const armpl_int_t *ltb, armpl_int_t *ipiv,
                      armpl_int_t *ipiv2, armpl_singlecomplex_t *b,
                      const armpl_int_t *ldb, armpl_singlecomplex_t *work,
                      const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, L is stored below (or above) the subdiaonal blocks, when UPLO is 'L' (or 'U').

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TB Output parameter.

TB is COMPLEX

TB is an array, dimension (LTB). On exit, details of the LU factorization of the band matrix.

LTB Input parameter.

The size of the array TB. LTB >= 4*N, internally

used to select NB such that LTB >= (3*NB+1)*N.

If LTB = -1, then a workspace query is assumed; the routine only calculates the optimal size of LTB, returns this value as the first entry of TB, and no error message related to LTB is issued by XERBLA.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

On exit, it contains the details of the interchanges, i.e., the row and column k of A were interchanged with the row and column IPIV(k).

IPIV2 Output parameter.

IPIV is INTEGER array, dimension (N)

On exit, it contains the details of the interchanges, i.e., the row and column k of T were interchanged with the row and column IPIV(k).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Output parameter.

WORK is COMPLEX workspace of size LWORK

LWORK Input parameter.

The size of WORK. LWORK >= N, internally used to select NB

such that LWORK >= N*NB.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, band LU factorization failed on i-th column

Related Information

For this routine in other precisions, please see *dsysv_aa_2stage*, *ssysv_aa_2stage* and *zsysv_aa_2stage*. It also exists with a native C interface as *LAPACKE_csysv_aa_2stage*.

4.16.24 csytf2_rk

CSYTF2_RK computes the factorization of a complex symmetric matrix A using the bounded Bunch-Kaufman (rook) diagonal pivoting method:

```
A = P*U*D*(U**T)*(P**T) or A = P*L*D*(L**T)*(P**T),
```

where U (or L) is unit upper (or lower) triangular matrix, U^T (or L^T) is the transpose of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the unblocked version of the algorithm, calling Level 2 BLAS. For more information see Further Details section.

Syntax

Fortran specification:

```
use armpl_library

subroutine csytf2_rk(UPLO, N, A, LDA, E, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void csytf2_rk_(const char *uplo, const armpl_int_t *n,
                armpl_singlecomplex_t *a, const armpl_int_t *lda,
                armpl_singlecomplex_t *e, armpl_int_t *ipiv,
                armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U': the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced.

If UPLO = 'L': the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, contains: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Output parameter.

E is COMPLEX

E is an array, dimension (N). On exit, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) is set to 0; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) is set to 0.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is set to 0 in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

IPIV describes the permutation matrix P in the factorization of matrix A as follows. The absolute value of IPIV(k) represents the index of row and column that were interchanged with the k-th row and column. The value of UPLO describes the order in which the interchanges were applied. Also, the sign of IPIV represents the block structure of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks which correspond to 1 or 2 interchanges at each factorization step. For more info see Further Details section.

If UPLO = 'U', (in factorization order, k decreases from N to 1): a) A single positive entry IPIV(k) > 0 means: D(k,k) is a 1-by-1 diagonal block. If IPIV(k) != k, rows and columns k and IPIV(k) were interchanged in the matrix A(1:N,1:N); If IPIV(k) = k, no interchange occurred.

b) A pair of consecutive negative entries IPIV(k) < 0 and IPIV(k-1) < 0 means: D(k-1:k,k-1:k) is a 2-by-2 diagonal block. (NOTE: negative entries in IPIV appear ONLY in pairs). 1) If -IPIV(k) != k, rows and columns k and -IPIV(k) were interchanged in the matrix A(1:N,1:N). If -IPIV(k) = k, no interchange occurred. 2) If -IPIV(k-1) != k-1, rows and columns k-1 and -IPIV(k-1) were interchanged in the matrix A(1:N,1:N). If -IPIV(k-1) = k-1, no interchange occurred.

c) In both cases a) and b), always ABS(IPIV(k)) <= k.

d) NOTE: Any entry IPIV(k) is always NONZERO on output.

If UPLO = 'L', (in factorization order, k increases from 1 to N): a) A single positive entry IPIV(k) > 0 means: D(k,k) is a 1-by-1 diagonal block. If IPIV(k) != k, rows and columns k and IPIV(k) were interchanged in the matrix A(1:N,1:N). If IPIV(k) = k, no interchange occurred.

b) A pair of consecutive negative entries IPIV(k) < 0 and IPIV(k+1) < 0 means: D(k:k+1,k:k+1) is a 2-by-2 diagonal block. (NOTE: negative entries in IPIV appear ONLY in pairs). 1) If -IPIV(k) != k, rows and columns k and -IPIV(k) were interchanged in the matrix A(1:N,1:N). If -IPIV(k) = k, no interchange occurred. 2) If -IPIV(k+1) != k+1, rows and columns k-1 and -IPIV(k-1) were interchanged in the matrix A(1:N,1:N). If -IPIV(k+1) = k+1, no interchange occurred.

c) In both cases a) and b), always ABS(IPIV(k)) >= k.

d) NOTE: Any entry IPIV(k) is always NONZERO on output.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit

< 0: If INFO = -k, the k-th argument had an illegal value

> 0: If INFO = k, the matrix A is singular, because: If UPLO = 'U': column k in the upper triangular part of A contains all zeros. If UPLO = 'L': column k in the lower triangular part of A contains all zeros.

Therefore D(k,k) is exactly zero, and superdiagonal elements of column k of U (or subdiagonal elements of column k of L) are all zeros. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

NOTE: INFO only stores the first occurrence of a singularity, any subsequent occurrence of singularity is not stored in INFO even though the factorization always completes.

Related Information

For this routine in other precisions, please see *dsytf2_rk*, *ssytf2_rk* and *zsytf2_rk*.

4.16.25 csytrf_aa_2stage

CSYTRF_AA_2STAGE computes the factorization of a complex symmetric matrix A using the Aasen's algorithm. The form of the factorization is

```
A = U*T*U**T  or  A = L*T*L**T
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and T is a complex symmetric band matrix with the bandwidth of NB (NB is internally selected and stored in TB(1), and T is LU factorized with partial pivoting).

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine csytrf_aa_2stage(UPLO, N, A, LDA, TB, LTB, IPIV, IPIV2, WORK,
                            LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void csytrf_aa_2stage_(const char *uplo, const armpl_int_t *n,
                       armpl_singlecomplex_t *a, const armpl_int_t *lda,
                       armpl_singlecomplex_t *tb, const armpl_int_t *ltb,
                       armpl_int_t *ipiv, armpl_int_t *ipiv2,
                       armpl_singlecomplex_t *work, const armpl_int_t *lwork,
                       armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, L is stored below (or above) the subdiaonal blocks, when UPLO is 'L' (or 'U').

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TB Output parameter.

TB is COMPLEX

TB is an array, dimension (LTB). On exit, details of the LU factorization of the band matrix.

LTB Input parameter.

The size of the array TB. LTB >= 4*N, internally

used to select NB such that LTB >= (3*NB+1)*N.

If LTB = -1, then a workspace query is assumed; the routine only calculates the optimal size of LTB, returns this value as the first entry of TB, and no error message related to LTB is issued by XERBLA.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

On exit, it contains the details of the interchanges, i.e., the row and column k of A were interchanged with the row and column IPIV(k).

IPIV2 Output parameter.

IPIV is INTEGER array, dimension (N)

On exit, it contains the details of the interchanges, i.e., the row and column k of T were interchanged with the row and column IPIV(k).

WORK Output parameter.

WORK is COMPLEX workspace of size LWORK

LWORK Input parameter.

The size of WORK. LWORK >= N, internally used to select NB

such that LWORK >= N*NB.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, band LU factorization failed on i-th column

Related Information

For this routine in other precisions, please see *dsytrf_aa_2stage*, *ssytrf_aa_2stage* and *zsytrf_aa_2stage*. It also exists with a native C interface as *LAPACKE_csytrf_aa_2stage*.

4.16.26 csytri_3x

CSYTRI_3X computes the inverse of a complex symmetric indefinite matrix A using the factorization computed by CSYTRF_RK or CSYTRF_BK:

```
A = P*U*D*(U**T)*(P**T) or A = P*L*D*(L**T)*(P**T),
```

where U (or L) is unit upper (or lower) triangular matrix, U^T (or L^T) is the transpose of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine csytri_3x(UPLO, N, A, LDA, E, IPIV, WORK, NB, INFO)
```

C specification:

```
#include "armpl.h"

void csytri_3x_(const char *uplo, const armpl_int_t *n,
                armpl_singlecomplex_t *a, const armpl_int_t *lda,
                const armpl_singlecomplex_t *e, const armpl_int_t *ipiv,
                armpl_singlecomplex_t *work, const armpl_int_t *nb,
                armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, diagonal of the block diagonal matrix D and factors U or L as computed by CSYTRF_RK and CSYTRF_BK: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D should be provided on entry in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

On exit, if INFO = 0, the symmetric inverse of the original matrix. If UPLO = 'U': the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; If UPLO = 'L': the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Input parameter.

E is COMPLEX

E is an array, dimension (N). On entry, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) not referenced; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) not referenced.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is not referenced in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CSYTRF_RK or CSYTRF_BK.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (N+NB+1,NB+3). .

NB Input parameter.

NB is INTEGER

Block size.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *dsytri_3x*, *ssytri_3x* and *zsytri_3x*.

4.16.27 csytrs_aa_2stage

CSYTRS_AA_2STAGE solves a system of linear equations A*X = B with a complex symmetric matrix A using the factorization A = U*T*U^T or A = L*T*L^T computed by CSYTRF_AA_2STAGE.

Syntax

Fortran specification:

```
use armpl_library

subroutine csytrs_aa_2stage(UPLO, N, NRHS, A, LDA, TB, LTB, IPIV, IPIV2, B,
                            LDB, INFO)
```

C specification:

```
#include "armpl.h"

void csytrs_aa_2stage_(const char *uplo, const armpl_int_t *n,
                       const armpl_int_t *nrhs,
                       const armpl_singlecomplex_t *a, const armpl_int_t *lda,
                       armpl_singlecomplex_t *tb, const armpl_int_t *ltb,
                       const armpl_int_t *ipiv, const armpl_int_t *ipiv2,
                       armpl_singlecomplex_t *b, const armpl_int_t *ldb,
                       armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*T*U^T ; = 'L': Lower triangular, form is A = L*T*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). Details of factors computed by CSYTRF_AA_2STAGE.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TB Output parameter.

TB is COMPLEX

TB is an array, dimension (LTB). Details of factors computed by CSYTRF_AA_2STAGE.

LTB Input parameter.

The size of the array TB. LTB >= 4*N.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges as computed by CSYTRF_AA_2STAGE.

IPIV2 Input parameter.

IPIV2 is INTEGER array, dimension (N)

Details of the interchanges as computed by CSYTRF_AA_2STAGE.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dsytrs_aa_2stage*, *ssytrs_aa_2stage* and *zsytrs_aa_2stage*. It also exists with a native C interface as *LAPACKE_csytrs_aa_2stage*.

4.16.28 ctrevc3

ctrevc3 computes some or all of the right and/or left eigenvectors of a complex upper triangular matrix T. Matrices of this type are produced by the Schur factorization of a complex general matrix: A = Q*T*Q^H , as computed by CHSEQR.

The right eigenvector x and the left eigenvector y of T corresponding to an eigenvalue w are defined by:

```
T*x = w*x,      (y**H)*T = w*(y**H)
```

where y^H denotes the conjugate transpose of the vector y. The eigenvalues are not input to this routine, but are read directly from the diagonal of T.

This routine returns the matrices X and/or Y of right and left eigenvectors of T, or the products Q*X and/or Q*Y, where Q is an input matrix. If Q is the unitary factor that reduces a matrix A to Schur form T, then Q*X and Q*Y are the matrices of right and left eigenvectors of A.

This uses a Level 3 BLAS version of the back transformation.

Syntax

Fortran specification:

```
use armpl_library

subroutine ctrevc3(SIDE, HOWMNY, SELECT, N, T, LDT, VL, LDVL, VR, LDVR, MM, M,
                   WORK, LWORK, RWORK, LRWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ctrevc3_(const char *side, const char *howmny, const armpl_int_t *select,
              const armpl_int_t *n, armpl_singlecomplex_t *t,
              const armpl_int_t *ldt, armpl_singlecomplex_t *vl,
              const armpl_int_t *ldvl, armpl_singlecomplex_t *vr,
              const armpl_int_t *ldvr, const armpl_int_t *mm, armpl_int_t *m,
              armpl_singlecomplex_t *work, const armpl_int_t *lwork,
              float *rwork, const armpl_int_t *lrwork, armpl_int_t *info,
              ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'R': compute right eigenvectors only; = 'L': compute left eigenvectors only; = 'B': compute both right and left eigenvectors.

HOWMNY Input parameter.

HOWMNY is CHARACTER*1

= 'A': compute all right and/or left eigenvectors; = 'B': compute all right and/or left eigenvectors, backtransformed using the matrices supplied in VR and/or VL; = 'S': compute selected right and/or left eigenvectors, as indicated by the logical array SELECT.

SELECT Input parameter.

SELECT is LOGICAL

SELECT is an array, dimension (N). If HOWMNY = 'S', SELECT specifies the eigenvectors to be computed. The eigenvector corresponding to the j-th eigenvalue is computed if SELECT(j) = .TRUE.. Not referenced if HOWMNY = 'A' or 'B'.

N Input parameter.

N is INTEGER

The order of the matrix T. N >= 0.

T Input and output parameter.

T is COMPLEX

T is an array, dimension (LDT, N). The upper triangular matrix T. T is modified, but restored on exit.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

VL Input and output parameter.

VL is COMPLEX

VL is an array, dimension (LDVL, MM). On entry, if SIDE = 'L' or 'B' and HOWMNY = 'B', VL must contain an N-by-N matrix Q (usually the unitary matrix Q of Schur vectors returned by CHSEQR). On exit, if SIDE = 'L' or 'B', VL contains: if HOWMNY = 'A', the matrix Y of left eigenvectors of T; if HOWMNY = 'B', the matrix Q*Y; if HOWMNY = 'S', the left eigenvectors of T specified by SELECT, stored consecutively in the columns of VL, in the same order as their eigenvalues. Not referenced if SIDE = 'R'.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the array VL. LDVL >= 1, and if SIDE = 'L' or 'B', LDVL >= N.

VR Input and output parameter.

VR is COMPLEX

VR is an array, dimension (LDVR, MM). On entry, if SIDE = 'R' or 'B' and HOWMNY = 'B', VR must contain an N-by-N matrix Q (usually the unitary matrix Q of Schur vectors returned by CHSEQR). On exit, if SIDE = 'R' or 'B', VR contains: if HOWMNY = 'A', the matrix X of right eigenvectors of T; if HOWMNY = 'B', the matrix Q*X; if HOWMNY = 'S', the right eigenvectors of T specified by SELECT, stored consecutively in the columns of VR, in the same order as their eigenvalues. Not referenced if SIDE = 'L'.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the array VR. LDVR >= 1, and if SIDE = 'R' or 'B', LDVR >= N.

MM Input parameter.

MM is INTEGER

The number of columns in the arrays VL and/or VR. MM >= M.

M Output parameter.

M is INTEGER

The number of columns in the arrays VL and/or VR actually used to store the eigenvectors. If HOWMNY = 'A' or 'B', M is set to N. Each selected eigenvector occupies one column.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (MAX(1, LWORK)) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of array WORK. LWORK >= max(1,2*N). For optimum performance, LWORK >= N + 2*N*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (LRWORK) .

LRWORK Input parameter.

LRWORK is INTEGER

The dimension of array RWORK. LRWORK >= max(1, N).

If LRWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the RWORK array, returns this value as the first entry of the RWORK array, and no error message related to LRWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dtrevc3*, *strevc3* and *ztrevc3*.

4.16.29 ctrtrs

ctrtrs solves a triangular system of the form

```
A * X = B,  A**T * X = B,  or  A**H * X = B,
```

where A is a triangular matrix of order N, and B is an N-by-NRHS matrix. A check is made to verify that A is nonsingular.

Syntax

Fortran specification:

```
use armpl_library

subroutine ctrtrs(UPLO, TRANS, DIAG, N, NRHS, A, LDA, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void ctrtrs_(const char *uplo, const char *trans, const char *diag,
             const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda,
             armpl_singlecomplex_t *b, const armpl_int_t *ldb,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose)

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, if INFO = 0, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element of A is zero, indicating that the matrix is singular and the solutions X have not been computed.

Related Information

For this routine in other precisions, please see *dtrtrs*, *strtrs* and *ztrtrs*. It also exists with a native C interface as *LAPACKE_ctrtrs*.

4.16.30 dcabs1

dcabs1 computes |Re(.)| + |Im(.)| of a double complex number

Syntax

Fortran specification:

```
use armpl_library

double precision function dcabs1(Z)
```

C specification:

```
#include "armpl.h"

double dcabs1_(const armpl_doublecomplex_t *z);
```

Parameters

Z Input parameter.

Z is COMPLEX*16

Related Information

For this routine in other precisions, please see *scabs1*.

4.16.31 dgetsls

dgetsls solves overdetermined or underdetermined real linear systems involving an M-by-N matrix A, using a tall skinny

The following options are provided:

* If TRANS = 'N' and m >= n: find the least squares solution of an overdetermined system, i.e., solve the least squares problem minimize || B - A*X ||.

* If TRANS = 'N' and m < n: find the minimum norm solution of an underdetermined system A * X = B.

* If TRANS = 'T' and m >= n: find the minimum norm solution of an undetermined system A^T * X = B.

* If TRANS = 'T' and m < n: find the least squares solution of an overdetermined system, i.e., solve the least squares problem minimize || B - A^T * X ||.

Several right hand side vectors b and solution vectors x can be handled in a single call; they are stored as the columns of the M-by-NRHS right hand side matrix B and the N-by-NRHS solution matrix X.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgetsls(TRANS, M, N, NRHS, A, LDA, B, LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgetsls_(const char *trans, const armpl_int_t *m, const armpl_int_t *n,
              const armpl_int_t *nrhs, double *a, const armpl_int_t *lda,
              double *b, const armpl_int_t *ldb, double *work,
              const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': the linear system involves A; = 'T': the linear system involves A^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >=0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, A is overwritten by details of its QR or LQ factorization as returned by DGEQR or DGELQ.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the matrix B of right hand side vectors, stored columnwise; B is M-by-NRHS if TRANS = 'N', or N-by-NRHS if TRANS = 'T'. On exit, if INFO = 0, B is overwritten by the solution vectors, stored columnwise: if TRANS = 'N' and m >= n, rows 1 to n of B contain the least squares solution vectors. if TRANS = 'N' and m < n, rows 1 to N of B contain the minimum norm solution vectors; if TRANS = 'T' and m >= n, rows 1 to M of B contain the minimum norm solution vectors; if TRANS = 'T' and m < n, rows 1 to M of B contain the least squares solution vectors.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= MAX(1, M, N).

WORK Output parameter.

(workspace) DOUBLE PRECISION

(workspace) is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) contains optimal (or either minimal or optimal, if query was assumed) LWORK. See LWORK for details.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If LWORK = -1 or -2, then a workspace query is assumed. If LWORK = -1, the routine calculates optimal size of WORK for the optimal performance and returns this value in WORK(1). If LWORK = -2, the routine calculates minimal size of WORK and returns this value in WORK(1).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element of the triangular factor of A is zero, so that A does not have full rank; the least squares solution could not be computed.

Related Information

For this routine in other precisions, please see *cgetsls*, *sgetsls* and *zgetsls*. It also exists with a native C interface as *LAPACKE_dgetsls*.

4.16.32 dlasyf_rk

DLASYF_RK computes a partial factorization of a real symmetric matrix A using the bounded Bunch-Kaufman (rook) diagonal pivoting method. The partial factorization has the form:

A = (I U12) (A11 0) (I 0) if UPLO = 'U', or:

```
( 0  U22 ) (  0   D  ) ( U12**T U22**T )
```

A = (L11 0) (D 0) ($L11^T$ $L21^T$) if UPLO = 'L',

```
( L21  I ) (  0  A22 ) (  0       I    )
```

where the order of D is at most NB. The actual order is returned in the argument KB, and is either NB or NB-1, or N if N <= NB.

DLASYF_RK is an auxiliary routine called by DSYTRF_RK. It uses blocked code (calling Level 3 BLAS) to update the submatrix A11 (if UPLO = 'U') or A22 (if UPLO = 'L').

Syntax

Fortran specification:

```
use armpl_library

subroutine dlasyf_rk(UPLO, N, NB, KB, A, LDA, E, IPIV, W, LDW, INFO)
```

C specification:

```
#include "armpl.h"

void dlasyf_rk_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nb,
                armpl_int_t *kb, double *a, const armpl_int_t *lda, double *e,
                armpl_int_t *ipiv, double *w, const armpl_int_t *ldw,
                armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NB Input parameter.

NB is INTEGER

The maximum number of columns of the matrix A that should be factored. NB should be at least 2 to allow for 2-by-2 pivot blocks.

KB Output parameter.

KB is INTEGER

The number of columns of A that were actually factored. KB is either NB-1 or NB, or N if N <= NB.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U': the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced.

If UPLO = 'L': the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, contains: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N). On exit, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) is set to 0; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) is set to 0.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is set to 0 in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

IPIV describes the permutation matrix P in the factorization of matrix A as follows. The absolute value of IPIV(k) represents the index of row and column that were interchanged with the k-th row and column. The value of UPLO describes the order in which the interchanges were applied. Also, the sign of IPIV represents the block structure of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks which correspond to 1 or 2 interchanges at each factorization step.

If UPLO = 'U', (in factorization order, k decreases from N to 1): a) A single positive entry IPIV(k) > 0 means: D(k,k) is a 1-by-1 diagonal block. If IPIV(k) != k, rows and columns k and IPIV(k) were interchanged in the submatrix A(1:N,N-KB+1:N); If IPIV(k) = k, no interchange occurred.

b) A pair of consecutive negative entries IPIV(k) < 0 and IPIV(k-1) < 0 means: D(k-1:k,k-1:k) is a 2-by-2 diagonal block. (NOTE: negative entries in IPIV appear ONLY in pairs). 1) If -IPIV(k) != k, rows and columns k and -IPIV(k) were interchanged in the matrix A(1:N,N-KB+1:N). If -IPIV(k) = k, no interchange occurred. 2) If -IPIV(k-1) != k-1, rows and columns k-1 and -IPIV(k-1) were interchanged in the submatrix A(1:N,N-KB+1:N). If -IPIV(k-1) = k-1, no interchange occurred.

c) In both cases a) and b) is always ABS(IPIV(k)) <= k.

d) NOTE: Any entry IPIV(k) is always NONZERO on output.

If UPLO = 'L', (in factorization order, k increases from 1 to N): a) A single positive entry IPIV(k) > 0 means: D(k,k) is a 1-by-1 diagonal block. If IPIV(k) != k, rows and columns k and IPIV(k) were interchanged in the submatrix A(1:N,1:KB). If IPIV(k) = k, no interchange occurred.

b) A pair of consecutive negative entries IPIV(k) < 0 and IPIV(k+1) < 0 means: D(k:k+1,k:k+1) is a 2-by-2 diagonal block. (NOTE: negative entries in IPIV appear ONLY in pairs). 1) If -IPIV(k) != k, rows and columns k and -IPIV(k) were interchanged in the submatrix A(1:N,1:KB). If -IPIV(k) = k, no interchange occurred. 2) If -IPIV(k+1) != k+1, rows and columns k-1 and -IPIV(k-1) were interchanged in the submatrix A(1:N,1:KB). If -IPIV(k+1) = k+1, no interchange occurred.

c) In both cases a) and b) is always ABS(IPIV(k)) >= k.

d) NOTE: Any entry IPIV(k) is always NONZERO on output.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (LDW, NB) .

LDW Input parameter.

LDW is INTEGER

The leading dimension of the array W. LDW >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit

< 0: If INFO = -k, the k-th argument had an illegal value

> 0: If INFO = k, the matrix A is singular, because: If UPLO = 'U': column k in the upper triangular part of A contains all zeros. If UPLO = 'L': column k in the lower triangular part of A contains all zeros.

Therefore D(k,k) is exactly zero, and superdiagonal elements of column k of U (or subdiagonal elements of column k of L) are all zeros. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

NOTE: INFO only stores the first occurrence of a singularity, any subsequent occurrence of singularity is not stored in INFO even though the factorization always completes.

Related Information

For this routine in other precisions, please see *clasyf_rk*, *slasyf_rk* and *zlasyf_rk*.

4.16.33 dsb2st_kernels

DSB2ST_KERNELS is an internal routine used by the DSYTRD_SB2ST subroutine.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsb2st_kernels(UPLO, WANTZ, TTYPE, ST, ED, SWEEP, N, NB, IB, A,
                          LDA, V, TAU, LDVT, WORK)
```

C specification:

```
#include "armpl.h"

void dsb2st_kernels_(const char *uplo, const armpl_int_t *wantz,
                     const armpl_int_t *ttype, const armpl_int_t *st,
                     const armpl_int_t *ed, const armpl_int_t *sweep,
                     const armpl_int_t *n, const armpl_int_t *nb,
                     const armpl_int_t *ib, double *a, const armpl_int_t *lda,
                     double *v, double *tau, const armpl_int_t *ldvt,
                     const double *work, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

WANTZ Input parameter.

WANTZ is LOGICAL which indicate if Eigenvalue are requested or both

Eigenvalue/Eigenvectors.

TTYPE Input parameter.

TTYPE is INTEGER

ST Input parameter.

ST is INTEGER

internal parameter for indices.

ED Input parameter.

ED is INTEGER

internal parameter for indices.

SWEEP Input parameter.

SWEEP is INTEGER

internal parameter for indices.

N Input parameter.

N is INTEGER. The order of the matrix A.

NB Input parameter.

NB is INTEGER. The size of the band.

IB Input parameter.

IB is INTEGER.

A Input and output parameter.

A is DOUBLE PRECISION array. A pointer to the matrix A.

LDA Input parameter.

LDA is INTEGER. The leading dimension of the matrix A.

V Output parameter.

V is DOUBLE PRECISION

V is an array, dimension 2*n if eigenvalues only are. requested or to be queried for vectors.

TAU Output parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (2*n).. The scalar factors of the Householder reflectors are stored in this array.

LDVT Input parameter.

LDVT is INTEGER.

WORK Input parameter.

WORK is DOUBLE PRECISION array. Workspace of size nb.

Related Information

For this routine in other precisions, please see *ssb2st_kernels*.

4.16.34 dsbev_2stage

DSBEV_2STAGE computes all the eigenvalues and, optionally, eigenvectors of a real symmetric band matrix A using the 2stage technique for the reduction to tridiagonal.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsbev_2stage(JOBZ, UPLO, N, KD, AB, LDAB, W, Z, LDZ, WORK, LWORK,
                        INFO)
```

C specification:

```
#include "armpl.h"

void dsbev_2stage_(const char *jobz, const char *uplo, const armpl_int_t *n,
                   const armpl_int_t *kd, double *ab, const armpl_int_t *ldab,
                   double *w, double *z, const armpl_int_t *ldz, double *work,
                   const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors. Not available in this release.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, AB is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the first superdiagonal and the diagonal of the tridiagonal matrix T are returned in rows KD and KD+1 of AB, and if UPLO = 'L', the diagonal and first subdiagonal of T are returned in the first two rows of AB.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD + 1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the orthonormal eigenvectors of the matrix A, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension LWORK. On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= 1, when N <= 1; otherwise If JOBZ = 'N' and N > 1, LWORK must be queried. LWORK = MAX(1, dimension) where dimension = (2KD+1)* N + KD*NTHREADS + N where KD is the size of the band. NTHREADS is the number of threads used when openMP compilation is enabled, otherwise =1. If JOBZ = 'V' and N > 1, LWORK must be queried. Not yet available.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *ssbev_2stage*. It also exists with a native C interface as *LAPACKE_dsbev_2stage*.

4.16.35 dsbevd_2stage

DSBEVD_2STAGE computes all the eigenvalues and, optionally, eigenvectors of a real symmetric band matrix A using the 2stage technique for the reduction to tridiagonal. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsbevd_2stage(JOBZ, UPLO, N, KD, AB, LDAB, W, Z, LDZ, WORK, LWORK,
                         IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dsbevd_2stage_(const char *jobz, const char *uplo, const armpl_int_t *n,
                    const armpl_int_t *kd, double *ab,
                    const armpl_int_t *ldab, double *w, double *z,
                    const armpl_int_t *ldz, double *work,
                    const armpl_int_t *lwork, armpl_int_t *iwork,
                    const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors. Not available in this release.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, AB is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the first superdiagonal and the diagonal of the tridiagonal matrix T are returned in rows KD and KD+1 of AB, and if UPLO = 'L', the diagonal and first subdiagonal of T are returned in the first two rows of AB.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD + 1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the orthonormal eigenvectors of the matrix A, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension LWORK. On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= 1, when N <= 1; otherwise If JOBZ = 'N' and N > 1, LWORK must be queried. LWORK = MAX(1, dimension) where dimension = (2KD+1)* N + KD*NTHREADS + N where KD is the size of the band. NTHREADS is the number of threads used when openMP compilation is enabled, otherwise =1. If JOBZ = 'V' and N > 1, LWORK must be queried. Not yet available.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. If JOBZ = 'N' or N <= 1, LIWORK must be at least 1. If JOBZ = 'V' and N > 2, LIWORK must be at least 3 + 5*N.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *ssbevd_2stage*. It also exists with a native C interface as *LAPACKE_dsbevd_2stage*.

4.16.36 dsbevx_2stage

DSBEVX_2STAGE computes selected eigenvalues and, optionally, eigenvectors of a real symmetric band matrix A using the 2stage technique for the reduction to tridiagonal. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsbevx_2stage(JOBZ, RANGE, UPLO, N, KD, AB, LDAB, Q, LDQ, VL, VU,
                         IL, IU, ABSTOL, M, W, Z, LDZ, WORK, LWORK, IWORK,
                         IFAIL, INFO)
```

C specification:

```
#include "armpl.h"

void dsbevx_2stage_(const char *jobz, const char *range, const char *uplo,
                    const armpl_int_t *n, const armpl_int_t *kd, double *ab,
                    const armpl_int_t *ldab, double *q,
                    const armpl_int_t *ldq, const double *vl,
                    const double *vu, const armpl_int_t *il,
                    const armpl_int_t *iu, const double *abstol,
                    armpl_int_t *m, double *w, double *z,
                    const armpl_int_t *ldz, double *work,
                    const armpl_int_t *lwork, armpl_int_t *iwork,
                    armpl_int_t *ifail, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors. Not available in this release.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found; = 'V': all eigenvalues in the half-open interval (VL,VU] will be found; = 'I': the IL-th through IU-th eigenvalues will be found.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, AB is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the first superdiagonal and the diagonal of the tridiagonal matrix T are returned in rows KD and KD+1 of AB, and if UPLO = 'L', the diagonal and first subdiagonal of T are returned in the first two rows of AB.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD + 1.

Q Output parameter.

Q is DOUBLE PRECISION

Q is an array, dimension (LDQ, N). If JOBZ = 'V', the N-by-N orthogonal matrix used in the reduction to tridiagonal form. If JOBZ = 'N', the array Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. If JOBZ = 'V', then LDQ >= max(1, N).

VL Input parameter.

VL is DOUBLE PRECISION

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is DOUBLE PRECISION

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is DOUBLE PRECISION

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing AB to tridiagonal form.

Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*DLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*DLAMCH('S').

See "Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy," by Demmel and Kahan, LAPACK Working Note #3.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). The first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). If an eigenvector fails to converge, then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL. If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= 1, when N <= 1; otherwise If JOBZ = 'N' and N > 1, LWORK must be queried. LWORK = MAX(1, 7*N, dimension) where dimension = (2KD+1)* N + KD*NTHREADS + 2*N where KD is the size of the band. NTHREADS is the number of threads used when openMP compilation is enabled, otherwise =1. If JOBZ = 'V' and N > 1, LWORK must be queried. Not yet available

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (5*N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (N)

If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, then i eigenvectors failed to converge. Their indices are stored in array IFAIL.

Related Information

For this routine in other precisions, please see *ssbevx_2stage*. It also exists with a native C interface as *LAPACKE_dsbevx_2stage*.

4.16.37 dsyconvf

If parameter WAY = 'C': DSYCONVF converts the factorization output format used in DSYTRF provided on entry in parameter A into the factorization output format used in DSYTRF_RK (or DSYTRF_BK) that is stored on exit in parameters A and E. It also coverts in place details of the intechanges stored in IPIV from the format used in DSYTRF into the format used in DSYTRF_RK (or DSYTRF_BK).

If parameter WAY = 'R': DSYCONVF performs the conversion in reverse direction, i.e. converts the factorization output format used in DSYTRF_RK (or DSYTRF_BK) provided on entry in parameters A and E into the factorization output format used in DSYTRF that is stored on exit in parameter A. It also coverts in place details of the intechanges stored in IPIV from the format used in DSYTRF_RK (or DSYTRF_BK) into the format used in DSYTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsyconvf(UPLO, WAY, N, A, LDA, E, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void dsyconvf_(const char *uplo, const char *way, const armpl_int_t *n,
               double *a, const armpl_int_t *lda, double *e,
               armpl_int_t *ipiv, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix A. = 'U': Upper triangular = 'L': Lower triangular

WAY Input parameter.

WAY is CHARACTER*1

= 'C': Convert = 'R': Revert

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). 1) If WAY ='C':

On entry, contains factorization details in format used in DSYTRF: a) all elements of the symmetric block diagonal matrix D on the diagonal of A and on superdiagonal (or subdiagonal) of A, and b) If UPLO = 'U': multipliers used to obtain factor U in the superdiagonal part of A. If UPLO = 'L': multipliers used to obtain factor L in the superdiagonal part of A.

On exit, contains factorization details in format used in DSYTRF_RK or DSYTRF_BK: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

2) If WAY = 'R':

On entry, contains factorization details in format used in DSYTRF_RK or DSYTRF_BK: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

On exit, contains factorization details in format used in DSYTRF: a) all elements of the symmetric block diagonal matrix D on the diagonal of A and on superdiagonal (or subdiagonal) of A, and b) If UPLO = 'U': multipliers used to obtain factor U in the superdiagonal part of A. If UPLO = 'L': multipliers used to obtain factor L in the superdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Input and output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N). 1) If WAY ='C':

On entry, just a workspace.

On exit, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) is set to 0; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) is set to 0.

2) If WAY = 'R':

On entry, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i),i=2:N, E(1) not referenced; If UPLO = 'L': E(i) = D(i+1,i),i=1:N-1, E(N) not referenced.

On exit, is not changed

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

1) If WAY ='C': On entry, details of the interchanges and the block structure of D in the format used in DSYTRF. On exit, details of the interchanges and the block structure of D in the format used in DSYTRF_RK (or DSYTRF_BK).

1) If WAY ='R': On entry, details of the interchanges and the block structure of D in the format used in DSYTRF_RK (or DSYTRF_BK). On exit, details of the interchanges and the block structure of D in the format used in DSYTRF.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csyconvf*, *ssyconvf* and *zsyconvf*.

4.16.38 dsyconvf_rook

If parameter WAY = 'C': DSYCONVF_ROOK converts the factorization output format used in DSYTRF_ROOK provided on entry in parameter A into the factorization output format used in DSYTRF_RK (or DSYTRF_BK) that is stored on exit in parameters A and E. IPIV format for DSYTRF_ROOK and DSYTRF_RK (or DSYTRF_BK) is the same and is not converted.

If parameter WAY = 'R': DSYCONVF_ROOK performs the conversion in reverse direction, i.e. converts the factorization output format used in DSYTRF_RK (or DSYTRF_BK) provided on entry in parameters A and E into the factorization output format used in DSYTRF_ROOK that is stored on exit in parameter A. IPIV format for DSYTRF_ROOK and DSYTRF_RK (or DSYTRF_BK) is the same and is not converted.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsyconvf_rook(UPLO, WAY, N, A, LDA, E, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void dsyconvf_rook_(const char *uplo, const char *way, const armpl_int_t *n,
                    double *a, const armpl_int_t *lda, double *e,
                    const armpl_int_t *ipiv, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix A. = 'U': Upper triangular = 'L': Lower triangular

WAY Input parameter.

WAY is CHARACTER*1

= 'C': Convert = 'R': Revert

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). 1) If WAY ='C':

On entry, contains factorization details in format used in DSYTRF_ROOK: a) all elements of the symmetric block diagonal matrix D on the diagonal of A and on superdiagonal (or subdiagonal) of A, and b) If UPLO = 'U': multipliers used to obtain factor U in the superdiagonal part of A. If UPLO = 'L': multipliers used to obtain factor L in the superdiagonal part of A.

On exit, contains factorization details in format used in DSYTRF_RK or DSYTRF_BK: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

2) If WAY = 'R':

On entry, contains factorization details in format used in DSYTRF_RK or DSYTRF_BK: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

On exit, contains factorization details in format used in DSYTRF_ROOK: a) all elements of the symmetric block diagonal matrix D on the diagonal of A and on superdiagonal (or subdiagonal) of A, and b) If UPLO = 'U': multipliers used to obtain factor U in the superdiagonal part of A. If UPLO = 'L': multipliers used to obtain factor L in the superdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Input and output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N). 1) If WAY ='C':

On entry, just a workspace.

On exit, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) is set to 0; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) is set to 0.

2) If WAY = 'R':

On entry, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i),i=2:N, E(1) not referenced; If UPLO = 'L': E(i) = D(i+1,i),i=1:N-1, E(N) not referenced.

On exit, is not changed

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

On entry, details of the interchanges and the block structure of D as determined: 1) by DSYTRF_ROOK, if WAY ='C'; 2) by DSYTRF_RK (or DSYTRF_BK), if WAY ='R'. The IPIV format is the same for all these routines.

On exit, is not changed.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csyconvf_rook*, *ssyconvf_rook* and *zsyconvf_rook*.

4.16.39 dsyev_2stage

DSYEV_2STAGE computes all eigenvalues and, optionally, eigenvectors of a real symmetric matrix A using the 2stage technique for the reduction to tridiagonal.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsyev_2stage(JOBZ, UPLO, N, A, LDA, W, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dsyev_2stage_(const char *jobz, const char *uplo, const armpl_int_t *n,
                   double *a, const armpl_int_t *lda, double *w, double *work,
                   const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors. Not available in this release.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading

N-by-N lower triangular part of A contains the lower triangular part of the matrix A. On exit, if JOBZ = 'V', then if INFO = 0, A contains the orthonormal eigenvectors of the matrix A. If JOBZ = 'N', then on exit the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension LWORK. On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= 1, when N <= 1; otherwise If JOBZ = 'N' and N > 1, LWORK must be queried. LWORK = MAX(1, dimension) where dimension = max(stage1,stage2) + (KD+1)* N + 2*N = N*KD + N*max(KD+1,FACTOPTNB) + max(2*KD*KD, KD*NTHREADS) + (KD+1)* N + 2*N where KD is the blocking size of the reduction, FACTOPTNB is the blocking used by the QR or LQ algorithm, usually FACTOPTNB=128 is a good choice NTHREADS is the number of threads used when openMP compilation is enabled, otherwise =1. If JOBZ = 'V' and N > 1, LWORK must be queried. Not yet available

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *ssyev_2stage*. It also exists with a native C interface as *LAPACKE_dsyev_2stage*.

4.16.40 dsyevd_2stage

DSYEVD_2STAGE computes all eigenvalues and, optionally, eigenvectors of a real symmetric matrix A using the 2stage technique for the reduction to tridiagonal. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsyevd_2stage(JOBZ, UPLO, N, A, LDA, W, WORK, LWORK, IWORK, LIWORK,
                         INFO)
```

C specification:

```
#include "armpl.h"

void dsyevd_2stage_(const char *jobz, const char *uplo, const armpl_int_t *n,
                    double *a, const armpl_int_t *lda, double *w,
                    double *work, const armpl_int_t *lwork,
                    armpl_int_t *iwork, const armpl_int_t *liwork,
                    armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors. Not available in this release.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A. On exit, if JOBZ = 'V', then if INFO = 0, A contains the orthonormal eigenvectors of the matrix A. If JOBZ = 'N', then on exit the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array,. dimension (LWORK) On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If N <= 1, LWORK must be at least 1. If JOBZ = 'N' and N > 1, LWORK must be queried. LWORK = MAX(1, dimension) where dimension = max(stage1,stage2) + (KD+1)* N +

2*N+1 = N*KD + N*max(KD+1,FACTOPTNB) + max(2*KD*KD, KD*NTHREADS) + (KD+1)* N + 2*N+1 where KD is the blocking size of the reduction, FACTOPTNB is the blocking used by the QR or LQ algorithm, usually FACTOPTNB=128 is a good choice NTHREADS is the number of threads used when openMP compilation is enabled, otherwise =1. If JOBZ = 'V' and N > 1, LWORK must be at least 1 + 6* N + 2*N**2.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. If N <= 1, LIWORK must be at least 1. If JOBZ = 'N' and N > 1, LIWORK must be at least 1. If JOBZ = 'V' and N > 1, LIWORK must be at least 3 + 5*N.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i and JOBZ = 'N', then the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero; if INFO = i and JOBZ = 'V', then the algorithm failed to compute an eigenvalue while working on the submatrix lying in rows and columns INFO/(N+1) through mod(INFO,N+1).

Related Information

For this routine in other precisions, please see *ssyevd_2stage*. It also exists with a native C interface as *LAPACKE_dsyevd_2stage*.

4.16.41 dsyevr_2stage

DSYEVR_2STAGE computes selected eigenvalues and, optionally, eigenvectors of a real symmetric matrix A using the 2stage technique for the reduction to tridiagonal. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

DSYEVR_2STAGE first reduces the matrix A to tridiagonal form T with a call to DSYTRD. Then, whenever possible, DSYEVR_2STAGE calls DSTEMR to compute the eigenspectrum using Relatively Robust Representations. DSTEMR computes eigenvalues by the dqds algorithm, while orthogonal eigenvectors are computed from various "good" L D L^T representations (also known as Relatively Robust Representations). Gram-Schmidt orthogonalization is avoided as far as possible. More specifically, the various steps of the algorithm are as follows.

For each unreduced block (submatrix) of T,

```
(a) Compute T - sigma I  = L D L^T, so that L and D
    define all the wanted eigenvalues to high relative accuracy.
    This means that small relative changes in the entries of D and L
    cause only small relative changes in the eigenvalues and
    eigenvectors. The standard (unfactored) representation of the
    tridiagonal matrix T does not have this property in general.
(b) Compute the eigenvalues to suitable accuracy.
    If the eigenvectors are desired, the algorithm attains full
```

(continues on next page)

(continued from previous page)

```
    accuracy of the computed eigenvalues only right before
    the corresponding vectors have to be computed, see steps c) and d).
(c) For each cluster of close eigenvalues, select a new
    shift close to the cluster, find a new factorization, and refine
    the shifted eigenvalues to suitable accuracy.
(d) For each eigenvalue with a large enough relative separation compute
    the corresponding eigenvector by forming a rank revealing twisted
    factorization. Go back to (c) for any clusters that remain.
```

The desired accuracy of the output can be specified by the input parameter ABSTOL.

For more details, see DSTEMR's documentation and: - Inderjit S. Dhillon and Beresford N. Parlett: "Multiple representations

```
 to compute orthogonal eigenvectors of symmetric tridiagonal matrices,"
Linear Algebra and its Applications, 387(1), pp. 1-28, August 2004.
```

- Inderjit Dhillon and Beresford Parlett: "Orthogonal Eigenvectors and

```
 Relative Gaps," SIAM Journal on Matrix Analysis and Applications, Vol. 25,
2004.  Also LAPACK Working Note 154.
```

- Inderjit Dhillon: "A new O(n^2) algorithm for the symmetric

```
 tridiagonal eigenvalue/eigenvector problem",
Computer Science Division Technical Report No. UCB/CSD-97-971,
UC Berkeley, May 1997.
```

Note 1 : DSYEVR_2STAGE calls DSTEMR when the full spectrum is requested on machines which conform to the ieee-754 floating point standard. DSYEVR_2STAGE calls DSTEBZ and SSTEIN on non-ieee machines and when partial spectrum requests are made.

Normal execution of DSTEMR may create NaNs and infinities and hence may abort due to a floating point exception in environments which do not handle NaNs and infinities in the ieee standard default manner.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsyevr_2stage(JOBZ, RANGE, UPLO, N, A, LDA, VL, VU, IL, IU, ABSTOL,
                         M, W, Z, LDZ, ISUPPZ, WORK, LWORK, IWORK, LIWORK,
                         INFO)
```

C specification:

```
#include "armpl.h"

void dsyevr_2stage_(const char *jobz, const char *range, const char *uplo,
                    const armpl_int_t *n, double *a, const armpl_int_t *lda,
                    const double *vl, const double *vu, const armpl_int_t *il,
                    const armpl_int_t *iu, const double *abstol,
                    armpl_int_t *m, double *w, double *z,
                    const armpl_int_t *ldz, armpl_int_t *isuppz, double *work,
                    const armpl_int_t *lwork, armpl_int_t *iwork,
                    const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors. Not available in this release.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found. = 'V': all eigenvalues in the half-open interval (VL,VU] will be found. = 'I': the IL-th through IU-th eigenvalues will be found. For RANGE = 'V' or 'I' and IU - IL < N - 1, DSTEBZ and DSTEIN are called

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A. On exit, the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

VL Input parameter.

VL is DOUBLE PRECISION

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is DOUBLE PRECISION

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is DOUBLE PRECISION

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing A to tridiagonal form.

See "Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy," by Demmel and Kahan, LAPACK Working Note #3.

If high relative accuracy is important, set ABSTOL to DLAMCH('Safe minimum'). Doing so will guarantee that eigenvalues are computed to high relative accuracy when possible in future releases. The current code does not make any guarantees about high relative accuracy, but future releases will. See J. Barlow and J. Demmel, "Computing Accurate Eigensystems of Scaled Diagonally Dominant Matrices", LAPACK Working Note #7, for a discussion of which matrices define their eigenvalues to high relative accuracy.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). The first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used. Supplying N columns is always safe.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

ISUPPZ Output parameter.

ISUPPZ is INTEGER array, dimension (2*max(1, M))

The support of the eigenvectors in Z, i.e., the indices indicating the nonzero elements in Z. The i-th eigenvector is nonzero only in elements ISUPPZ(2*i-1) through ISUPPZ(2*i). This is an output of DSTEMR (tridiagonal matrix). The support of the eigenvectors of A is typically 1:N because of the orthogonal transformations applied by DORMTR. Implemented only for RANGE = 'A' or 'I' and IU - IL = N - 1

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If JOBZ = 'N' and N > 1, LWORK must be queried. LWORK = MAX(1, 26*N, dimension) where dimension = max(stage1,stage2) + (KD+1)* N + 5*N = N*KD +

N*max(KD+1,FACTOPTNB) + max(2*KD*KD, KD*NTHREADS) + (KD+1)* N + 5*N where KD is the blocking size of the reduction, FACTOPTNB is the blocking used by the QR or LQ algorithm, usually FACTOPTNB=128 is a good choice NTHREADS is the number of threads used when openMP compilation is enabled, otherwise =1. If JOBZ = 'V' and N > 1, LWORK must be queried. Not yet available

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. LIWORK >= max(1,10*N).

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the IWORK array, returns this value as the first entry of the IWORK array, and no error message related to LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: Internal error

Related Information

For this routine in other precisions, please see *ssyevr_2stage*. It also exists with a native C interface as *LAPACKE_dsyevr_2stage*.

4.16.42 dsyevx_2stage

DSYEVX_2STAGE computes selected eigenvalues and, optionally, eigenvectors of a real symmetric matrix A using the 2stage technique for the reduction to tridiagonal. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsyevx_2stage(JOBZ, RANGE, UPLO, N, A, LDA, VL, VU, IL, IU, ABSTOL,
                         M, W, Z, LDZ, WORK, LWORK, IWORK, IFAIL, INFO)
```

C specification:

```
#include "armpl.h"

void dsyevx_2stage_(const char *jobz, const char *range, const char *uplo,
                    const armpl_int_t *n, double *a, const armpl_int_t *lda,
                    const double *vl, const double *vu, const armpl_int_t *il,
                    const armpl_int_t *iu, const double *abstol,
                    armpl_int_t *m, double *w, double *z,
                    const armpl_int_t *ldz, double *work,
                    const armpl_int_t *lwork, armpl_int_t *iwork,
                    armpl_int_t *ifail, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors. Not available in this release.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found. = 'V': all eigenvalues in the half-open interval (VL,VU] will be found. = 'I': the IL-th through IU-th eigenvalues will be found.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A. On exit, the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

VL Input parameter.

VL is DOUBLE PRECISION

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is DOUBLE PRECISION

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is DOUBLE PRECISION

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing A to tridiagonal form.

Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*DLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*DLAMCH('S').

See "Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy," by Demmel and Kahan, LAPACK Working Note #3.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). On normal exit, the first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). If an eigenvector fails to converge, then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL. If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= 1, when N <= 1; otherwise If JOBZ = 'N' and N > 1, LWORK must be queried. LWORK = MAX(1, 8*N, dimension) where dimension = max(stage1,stage2) + (KD+1)* N + 3*N = N*KD + N*max(KD+1,FACTOPTNB) + max(2*KD*KD, KD*NTHREADS) + (KD+1)* N + 3*N where KD is the blocking size of the reduction, FACTOPTNB is the blocking used by the QR or LQ algorithm, usually FACTOPTNB=128 is a good choice NTHREADS is the number of threads used when openMP compilation is enabled, otherwise =1. If JOBZ = 'V' and N > 1, LWORK must be queried. Not yet available

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (5*N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (N)

If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, then i eigenvectors failed to converge. Their indices are stored in array IFAIL.

Related Information

For this routine in other precisions, please see *ssyevx_2stage*. It also exists with a native C interface as *LAPACKE_dsyevx_2stage*.

4.16.43 dsygv_2stage

DSYGV_2STAGE computes all the eigenvalues, and optionally, the eigenvectors of a real generalized symmetric-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x. Here A and B are assumed to be symmetric and B is also positive definite. This routine use the 2stage technique for the reduction to tridiagonal which showed higher performance on recent architecture and for large sizes N>2000.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsygv_2stage(ITYPE, JOBZ, UPLO, N, A, LDA, B, LDB, W, WORK, LWORK,
                        INFO)
```

C specification:

```
#include "armpl.h"

void dsygv_2stage_(const armpl_int_t *itype, const char *jobz,
                   const char *uplo, const armpl_int_t *n, double *a,
                   const armpl_int_t *lda, double *b, const armpl_int_t *ldb,
                   double *w, double *work, const armpl_int_t *lwork,
                   armpl_int_t *info, ... );
```

Parameters

ITYPE Input parameter.

ITYPE is INTEGER

Specifies the problem type to be solved: = 1: A*x = (lambda)*B*x = 2: A*B*x = (lambda)*x = 3: B*A*x = (lambda)*x

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors. Not available in this release.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangles of A and B are stored; = 'L': Lower triangles of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A.

On exit, if JOBZ = 'V', then if INFO = 0, A contains the matrix Z of eigenvectors. The eigenvectors are normalized as follows: if ITYPE = 1 or 2, Z^T *B*Z = I; if ITYPE = 3, Z^T *inv(B)*Z = I. If JOBZ = 'N', then on exit the upper triangle (if UPLO='U') or the lower triangle (if UPLO='L') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). On entry, the symmetric positive definite matrix B. If UPLO = 'U', the leading N-by-N upper triangular part of B contains the upper triangular part of the matrix B. If UPLO = 'L', the leading N-by-N lower triangular part of B contains the lower triangular part of the matrix B.

On exit, if INFO <= N, the part of B containing the matrix is overwritten by the triangular factor U or L from the Cholesky factorization B = U^T *U or B = L*L^T .

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= 1, when N <= 1; otherwise If JOBZ = 'N' and N > 1, LWORK must be queried. LWORK = MAX(1, dimension) where dimension = max(stage1,stage2) + (KD+1)* N + 2*N = N*KD + N*max(KD+1,FACTOPTNB) + max(2*KD*KD, KD*NTHREADS) + (KD+1)* N + 2*N where

KD is the blocking size of the reduction, FACTOPTNB is the blocking used by the QR or LQ algorithm, usually FACTOPTNB=128 is a good choice NTHREADS is the number of threads used when openMP compilation is enabled, otherwise =1. If JOBZ = 'V' and N > 1, LWORK must be queried. Not yet available

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: DPOTRF or DSYEV returned an error code: <= N: if INFO = i, DSYEV failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero; > N: if INFO = N + i, for 1 <= i <= N, then the leading minor of order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *ssygv_2stage*. It also exists with a native C interface as *LAPACKE_dsygv_2stage*.

4.16.44 dsysv_aa_2stage

DSYSV_AA_2STAGE computes the solution to a real system of linear equations

```
A * X = B,
```

where A is an N-by-N symmetric matrix and X and B are N-by-NRHS matrices.

Aasen's 2-stage algorithm is used to factor A as

```
A = U * T * U**T,  if UPLO = 'U', or
A = L * T * L**T,  if UPLO = 'L',
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and T is symmetric and band. The matrix T is then LU-factored with partial pivoting. The factored form of A is then used to solve the system of equations A * X = B.

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsysv_aa_2stage(UPLO, N, NRHS, A, LDA, TB, LTB, IPIV, IPIV2, B,
                           LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dsysv_aa_2stage_(const char *uplo, const armpl_int_t *n,
                      const armpl_int_t *nrhs, double *a,
                      const armpl_int_t *lda, double *tb,
                      const armpl_int_t *ltb, armpl_int_t *ipiv,
                      armpl_int_t *ipiv2, double *b, const armpl_int_t *ldb,
```

(continues on next page)

(continued from previous page)

```
                        double *work, const armpl_int_t *lwork,
                        armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, L is stored below (or above) the subdiaonal blocks, when UPLO is 'L' (or 'U').

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TB Output parameter.

TB is DOUBLE PRECISION

TB is an array, dimension (LTB). On exit, details of the LU factorization of the band matrix.

LTB Input parameter.

The size of the array TB. LTB >= 4*N, internally

used to select NB such that LTB >= (3*NB+1)*N.

If LTB = -1, then a workspace query is assumed; the routine only calculates the optimal size of LTB, returns this value as the first entry of TB, and no error message related to LTB is issued by XERBLA.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

On exit, it contains the details of the interchanges, i.e., the row and column k of A were interchanged with the row and column IPIV(k).

IPIV2 Output parameter.

IPIV is INTEGER array, dimension (N)

On exit, it contains the details of the interchanges, i.e., the row and column k of T were interchanged with the row and column IPIV(k).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION workspace of size LWORK

LWORK Input parameter.

The size of WORK. LWORK >= N, internally used to select NB

such that LWORK >= N*NB.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, band LU factorization failed on i-th column

Related Information

For this routine in other precisions, please see *csysv_aa_2stage*, *ssysv_aa_2stage* and *zsysv_aa_2stage*. It also exists with a native C interface as *LAPACKE_dsysv_aa_2stage*.

4.16.45 dsytf2_rk

DSYTF2_RK computes the factorization of a real symmetric matrix A using the bounded Bunch-Kaufman (rook) diagonal pivoting method:

```
A = P*U*D*(U**T)*(P**T) or A = P*L*D*(L**T)*(P**T),
```

where U (or L) is unit upper (or lower) triangular matrix, U^T (or L^T) is the transpose of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the unblocked version of the algorithm, calling Level 2 BLAS. For more information see Further Details section.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsytf2_rk(UPLO, N, A, LDA, E, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void dsytf2_rk_(const char *uplo, const armpl_int_t *n, double *a,
                const armpl_int_t *lda, double *e, armpl_int_t *ipiv,
                armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U': the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced.

If UPLO = 'L': the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, contains: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N). On exit, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) is set to 0; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) is set to 0.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is set to 0 in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

IPIV describes the permutation matrix P in the factorization of matrix A as follows. The absolute value of IPIV(k) represents the index of row and column that were interchanged with the k-th row and column. The value of UPLO describes the order in which the interchanges were applied. Also, the sign of IPIV represents the block structure of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks which correspond to 1 or 2 interchanges at each factorization step. For more info see Further Details section.

If UPLO = 'U', (in factorization order, k decreases from N to 1): a) A single positive entry IPIV(k) > 0 means: D(k,k) is a 1-by-1 diagonal block. If IPIV(k) != k, rows and columns k and IPIV(k) were interchanged in the matrix A(1:N,1:N); If IPIV(k) = k, no interchange occurred.

b) A pair of consecutive negative entries IPIV(k) < 0 and IPIV(k-1) < 0 means: D(k-1:k,k-1:k) is a 2-by-2 diagonal block. (NOTE: negative entries in IPIV appear ONLY in pairs). 1) If -IPIV(k) != k, rows and columns k and -IPIV(k) were interchanged in the matrix A(1:N,1:N). If -IPIV(k) = k, no interchange occurred. 2) If -IPIV(k-1) != k-1, rows and columns k-1 and -IPIV(k-1) were interchanged in the matrix A(1:N,1:N). If -IPIV(k-1) = k-1, no interchange occurred.

c) In both cases a) and b), always ABS(IPIV(k)) <= k.

d) NOTE: Any entry IPIV(k) is always NONZERO on output.

If UPLO = 'L', (in factorization order, k increases from 1 to N): a) A single positive entry IPIV(k) > 0 means: D(k,k) is a 1-by-1 diagonal block. If IPIV(k) != k, rows and columns k and IPIV(k) were interchanged in the matrix A(1:N,1:N). If IPIV(k) = k, no interchange occurred.

b) A pair of consecutive negative entries IPIV(k) < 0 and IPIV(k+1) < 0 means: D(k:k+1,k:k+1) is a 2-by-2 diagonal block. (NOTE: negative entries in IPIV appear ONLY in pairs). 1) If -IPIV(k) != k, rows and columns k and -IPIV(k) were interchanged in the matrix A(1:N,1:N). If -IPIV(k) = k, no interchange occurred. 2) If -IPIV(k+1) != k+1, rows and columns k-1 and -IPIV(k-1) were interchanged in the matrix A(1:N,1:N). If -IPIV(k+1) = k+1, no interchange occurred.

c) In both cases a) and b), always ABS(IPIV(k)) >= k.

d) NOTE: Any entry IPIV(k) is always NONZERO on output.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit

< 0: If INFO = -k, the k-th argument had an illegal value

> 0: If INFO = k, the matrix A is singular, because: If UPLO = 'U': column k in the upper triangular part of A contains all zeros. If UPLO = 'L': column k in the lower triangular part of A contains all zeros.

Therefore D(k,k) is exactly zero, and superdiagonal elements of column k of U (or subdiagonal elements of column k of L) are all zeros. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

NOTE: INFO only stores the first occurrence of a singularity, any subsequent occurrence of singularity is not stored in INFO even though the factorization always completes.

Related Information

For this routine in other precisions, please see *csytf2_rk*, *ssytf2_rk* and *zsytf2_rk*.

4.16.46 dsytrd_2stage

DSYTRD_2STAGE reduces a real symmetric matrix A to real symmetric tridiagonal form T by a orthogonal similarity transformation: $Q1^T$ $Q2^T$ * A * Q2 * Q1 = T.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsytrd_2stage(VECT, UPLO, N, A, LDA, D, E, TAU, HOUS2, LHOUS2,
                         WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dsytrd_2stage_(const char *vect, const char *uplo, const armpl_int_t *n,
                    double *a, const armpl_int_t *lda, double *d, double *e,
                    double *tau, double *hous2, const armpl_int_t *lhous2,
                    double *work, const armpl_int_t *lwork, armpl_int_t *info,
                    ... );
```

Parameters

VECT Input parameter.

VECT is CHARACTER*1

= 'N': No need for the Housholder representation, in particular for the second stage (Band to tridiagonal) and thus LHOUS2 is of size max(1, 4*N); = 'V': the Householder representation is needed to either generate Q1 Q2 or to apply Q1 Q2, then LHOUS2 is to be queried and computed. (NOT AVAILABLE IN THIS RELEASE).

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, if UPLO = 'U', the band superdiagonal of A are overwritten by the corresponding elements of the internal band-diagonal matrix AB, and the elements above the KD superdiagonal, with the array TAU, represent the orthogonal matrix Q1 as a product of elementary reflectors; if UPLO = 'L', the diagonal and band subdiagonal of A are over- written by the corresponding elements of the internal band-diagonal matrix AB, and the elements below the KD subdiagonal, with the array TAU, represent the orthogonal matrix Q1 as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

D Output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The diagonal elements of the tridiagonal matrix T.

E Output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N-1). The off-diagonal elements of the tridiagonal matrix T.

TAU Output parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (N-KD). The scalar factors of the elementary reflectors of the first stage (see Further Details).

HOUS2 Output parameter.

HOUS2 is DOUBLE PRECISION

HOUS2 is an array, dimension LHOUS2, that. store the Householder representation of the stage2 band to tridiagonal.

LHOUS2 Input parameter.

LHOUS2 is INTEGER

The dimension of the array HOUS2. LHOUS2 = MAX(1, dimension) If LWORK = -1, or LHOUS2=-1, then a query is assumed; the routine only calculates the optimal size of the HOUS2 array, returns this value as the first entry of the HOUS2 array, and no error message related to LHOUS2 is issued by XERBLA. LHOUS2 = MAX(1, dimension) where dimension = 4* N if VECT='N' not available now if VECT='H'

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK = MAX(1, dimension) If LWORK = -1, or LHOUS2=-1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA. LWORK = MAX(1, dimension) where dimension = max(stage1,stage2) + (KD+1)*N = N*KD + N*max(KD+1,FACTOPTNB) + max(2*KD*KD, KD*NTHREADS) + (KD+1)* N where KD is the blocking size of the reduction, FACTOPTNB is the blocking used by the QR or LQ algorithm, usually FACTOPTNB=128 is a good choice NTHREADS is the number of threads used when openMP compilation is enabled, otherwise =1.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ssytrd_2stage*.

4.16.47 dsytrd_sy2sb

DSYTRD_SY2SB reduces a real symmetric matrix A to real symmetric band-diagonal form AB by a orthogonal similarity transformation: Q^T * A * Q = AB.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsytrd_sy2sb(UPLO, N, KD, A, LDA, AB, LDAB, TAU, WORK, LWORK,
                        INFO)
```

C specification:

```
#include "armpl.h"

void dsytrd_sy2sb_(const char *uplo, const armpl_int_t *n,
                   const armpl_int_t *kd, double *a, const armpl_int_t *lda,
                   double *ab, const armpl_int_t *ldab, double *tau,
                   double *work, const armpl_int_t *lwork, armpl_int_t *info,
                   ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the reduced matrix if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0. The reduced matrix is stored in the array AB.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, if UPLO = 'U', the diagonal and first superdiagonal of A are overwritten by the corresponding elements of the tridiagonal matrix T, and the elements above the first superdiagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors; if UPLO = 'L', the diagonal and first subdiagonal of A are over- written by the corresponding elements of the tridiagonal matrix T, and the elements below the first subdiagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AB Output parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). On exit, the upper or lower triangle of the symmetric band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

TAU Output parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (N-KD). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (LWORK). On exit, if INFO = 0, or if LWORK=-1, WORK(1) returns the size of LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK which should be calculated by a workspace query. LWORK = MAX(1, LWORK_QUERY) If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA. LWORK_QUERY = N* KD + N*max(KD,FACTOPTNB) + 2*KD*KD where FACTOPTNB is the blocking used by the QR or LQ algorithm, usually FACTOPTNB=128 is a good choice otherwise putting LWORK=-1 will provide the size of WORK.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ssytrd_sy2sb*.

4.16.48 dsytrf_aa_2stage

DSYTRF_AA_2STAGE computes the factorization of a real symmetric matrix A using the Aasen's algorithm. The form of the factorization is

```
A = U*T*U**T  or  A = L*T*L**T
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and T is a symmetric band matrix with the bandwidth of NB (NB is internally selected and stored in TB(1), and T is LU factorized with partial pivoting).

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsytrf_aa_2stage(UPLO, N, A, LDA, TB, LTB, IPIV, IPIV2, WORK,
                            LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dsytrf_aa_2stage_(const char *uplo, const armpl_int_t *n, double *a,
                       const armpl_int_t *lda, double *tb,
                       const armpl_int_t *ltb, armpl_int_t *ipiv,
                       armpl_int_t *ipiv2, double *work,
                       const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, L is stored below (or above) the subdiaonal blocks, when UPLO is 'L' (or 'U').

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TB Output parameter.

TB is DOUBLE PRECISION

TB is an array, dimension (LTB). On exit, details of the LU factorization of the band matrix.

LTB Input parameter.

The size of the array TB. LTB >= 4*N, internally

used to select NB such that LTB >= (3*NB+1)*N.

If LTB = -1, then a workspace query is assumed; the routine only calculates the optimal size of LTB, returns this value as the first entry of TB, and no error message related to LTB is issued by XERBLA.

WORK Output parameter.

WORK is DOUBLE PRECISION workspace of size LWORK

LWORK Input parameter.

The size of WORK. LWORK >= N, internally used to select NB

such that LWORK >= N*NB.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

On exit, it contains the details of the interchanges, i.e., the row and column k of A were interchanged with the row and column IPIV(k).

IPIV2 Output parameter.

IPIV is INTEGER array, dimension (N)

On exit, it contains the details of the interchanges, i.e., the row and column k of T were interchanged with the row and column IPIV(k).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, band LU factorization failed on i-th column

Related Information

For this routine in other precisions, please see *csytrf_aa_2stage*, *ssytrf_aa_2stage* and *zsytrf_aa_2stage*. It also exists with a native C interface as *LAPACKE_dsytrf_aa_2stage*.

4.16.49 dsytri_3x

DSYTRI_3X computes the inverse of a real symmetric indefinite matrix A using the factorization computed by DSYTRF_RK or DSYTRF_BK:

```
A = P*U*D*(U**T)*(P**T) or A = P*L*D*(L**T)*(P**T),
```

where U (or L) is unit upper (or lower) triangular matrix, U^T (or L^T) is the transpose of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsytri_3x(UPLO, N, A, LDA, E, IPIV, WORK, NB, INFO)
```

C specification:

```
#include "armpl.h"

void dsytri_3x_(const char *uplo, const armpl_int_t *n, double *a,
                const armpl_int_t *lda, const double *e,
                const armpl_int_t *ipiv, double *work, const armpl_int_t *nb,
                armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, diagonal of the block diagonal matrix D and factors U or L as computed by DSYTRF_RK and DSYTRF_BK: a) ONLY diagonal elements of the symmetric block diagonal

matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D should be provided on entry in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

On exit, if INFO = 0, the symmetric inverse of the original matrix. If UPLO = 'U': the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; If UPLO = 'L': the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Input parameter.

E is DOUBLE PRECISION

E is an array, dimension (N). On entry, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) not referenced; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) not referenced.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is not referenced in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by DSYTRF_RK or DSYTRF_BK.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (N+NB+1,NB+3). .

NB Input parameter.

NB is INTEGER

Block size.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *csytri_3x*, *ssytri_3x* and *zsytri_3x*.

4.16.50 dsytrs_aa_2stage

DSYTRS_AA_2STAGE solves a system of linear equations A*X = B with a real symmetric matrix A using the factorization A = U*T*U^T or A = L*T*L^T computed by DSYTRF_AA_2STAGE.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsytrs_aa_2stage(UPLO, N, NRHS, A, LDA, TB, LTB, IPIV, IPIV2, B,
                            LDB, INFO)
```

C specification:

```
#include "armpl.h"

void dsytrs_aa_2stage_(const char *uplo, const armpl_int_t *n,
                       const armpl_int_t *nrhs, const double *a,
                       const armpl_int_t *lda, double *tb,
                       const armpl_int_t *ltb, const armpl_int_t *ipiv,
                       const armpl_int_t *ipiv2, double *b,
                       const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*T*U^T ; = 'L': Lower triangular, form is A = L*T*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). Details of factors computed by DSYTRF_AA_2STAGE.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TB Output parameter.

TB is DOUBLE PRECISION

TB is an array, dimension (LTB). Details of factors computed by DSYTRF_AA_2STAGE.

LTB Input parameter.

The size of the array TB. LTB >= 4*N.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges as computed by DSYTRF_AA_2STAGE.

IPIV2 Input parameter.

IPIV2 is INTEGER array, dimension (N)

Details of the interchanges as computed by DSYTRF_AA_2STAGE.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csytrs_aa_2stage*, *ssytrs_aa_2stage* and *zsytrs_aa_2stage*. It also exists with a native C interface as *LAPACKE_dsytrs_aa_2stage*.

4.16.51 dtrevc3

dtrevc3 computes some or all of the right and/or left eigenvectors of a real upper quasi-triangular matrix T. Matrices of this type are produced by the Schur factorization of a real general matrix: A = Q*T*Q^T , as computed by DHSEQR.

The right eigenvector x and the left eigenvector y of T corresponding to an eigenvalue w are defined by:

```
T*x = w*x,      (y**H)*T = w*(y**H)
```

where y^H denotes the conjugate transpose of y. The eigenvalues are not input to this routine, but are read directly from the diagonal blocks of T.

This routine returns the matrices X and/or Y of right and left eigenvectors of T, or the products Q*X and/or Q*Y, where Q is an input matrix. If Q is the orthogonal factor that reduces a matrix A to Schur form T, then Q*X and Q*Y are the matrices of right and left eigenvectors of A.

This uses a Level 3 BLAS version of the back transformation.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtrevc3(SIDE, HOWMNY, SELECT, N, T, LDT, VL, LDVL, VR, LDVR, MM, M,
                   WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dtrevc3_(const char *side, const char *howmny, armpl_int_t *select,
              const armpl_int_t *n, const double *t, const armpl_int_t *ldt,
              double *vl, const armpl_int_t *ldvl, double *vr,
              const armpl_int_t *ldvr, const armpl_int_t *mm, armpl_int_t *m,
              double *work, const armpl_int_t *lwork, armpl_int_t *info,
              ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'R': compute right eigenvectors only; = 'L': compute left eigenvectors only; = 'B': compute both right and left eigenvectors.

HOWMNY Input parameter.

HOWMNY is CHARACTER*1

= 'A': compute all right and/or left eigenvectors; = 'B': compute all right and/or left eigenvectors, backtransformed by the matrices in VR and/or VL; = 'S': compute selected right and/or left eigenvectors, as indicated by the logical array SELECT.

SELECT Input and output parameter.

SELECT is LOGICAL

SELECT is an array, dimension (N). If HOWMNY = 'S', SELECT specifies the eigenvectors to be computed. If w(j) is a real eigenvalue, the corresponding real eigenvector is computed if SELECT(j) is .TRUE.. If w(j) and w(j+1) are the real and imaginary parts of a complex eigenvalue, the corresponding complex eigenvector is computed if either SELECT(j) or SELECT(j+1) is .TRUE., and on exit SELECT(j) is set to .TRUE. and SELECT(j+1) is set to .FALSE.. Not referenced if HOWMNY = 'A' or 'B'.

N Input parameter.

N is INTEGER

The order of the matrix T. N >= 0.

T Input parameter.

T is DOUBLE PRECISION

T is an array, dimension (LDT, N). The upper quasi-triangular matrix T in Schur canonical form.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

VL Input and output parameter.

VL is DOUBLE PRECISION

VL is an array, dimension (LDVL, MM). On entry, if SIDE = 'L' or 'B' and HOWMNY = 'B', VL must contain an N-by-N matrix Q (usually the orthogonal matrix Q of Schur vectors returned by DHSEQR). On exit, if SIDE = 'L' or 'B', VL contains: if HOWMNY = 'A', the matrix Y of left eigenvectors of T; if HOWMNY = 'B', the matrix Q*Y; if HOWMNY = 'S', the left eigenvectors of T specified by SELECT, stored consecutively in the columns of VL, in the same order as their eigenvalues. A complex eigenvector corresponding to a complex eigenvalue is stored in two consecutive columns, the first holding the real part, and the second the imaginary part. Not referenced if SIDE = 'R'.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the array VL. LDVL >= 1, and if SIDE = 'L' or 'B', LDVL >= N.

VR Input and output parameter.

VR is DOUBLE PRECISION

VR is an array, dimension (LDVR, MM). On entry, if SIDE = 'R' or 'B' and HOWMNY = 'B', VR must contain an N-by-N matrix Q (usually the orthogonal matrix Q of Schur vectors returned by DHSEQR). On exit, if SIDE = 'R' or 'B', VR contains: if HOWMNY = 'A', the matrix X of right eigenvectors of T; if HOWMNY = 'B', the matrix Q*X; if HOWMNY = 'S', the right eigenvectors of T specified by SELECT, stored consecutively

in the columns of VR, in the same order as their eigenvalues. A complex eigenvector corresponding to a complex eigenvalue is stored in two consecutive columns, the first holding the real part and the second the imaginary part. Not referenced if SIDE = 'L'.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the array VR. LDVR >= 1, and if SIDE = 'R' or 'B', LDVR >= N.

MM Input parameter.

MM is INTEGER

The number of columns in the arrays VL and/or VR. MM >= M.

M Output parameter.

M is INTEGER

The number of columns in the arrays VL and/or VR actually used to store the eigenvectors. If HOWMNY = 'A' or 'B', M is set to N. Each selected real eigenvector occupies one column and each selected complex eigenvector occupies two columns.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of array WORK. LWORK >= max(1,3*N). For optimum performance, LWORK >= N + 2*N*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctrevc3*, *strevc3* and *ztrevc3*.

4.16.52 dtrtrs

dtrtrs solves a triangular system of the form

```
A * X = B  or  A**T * X = B,
```

where A is a triangular matrix of order N, and B is an N-by-NRHS matrix. A check is made to verify that A is nonsingular.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtrtrs(UPLO, TRANS, DIAG, N, NRHS, A, LDA, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void dtrtrs_(const char *uplo, const char *trans, const char *diag,
             const armpl_int_t *n, const armpl_int_t *nrhs, const double *a,
             const armpl_int_t *lda, double *b, const armpl_int_t *ldb,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose = Transpose)

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). The triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, if INFO = 0, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element of A is zero, indicating that the matrix is singular and the solutions X have not been computed.

Related Information

For this routine in other precisions, please see *ctrtrs*, *strtrs* and *ztrtrs*. It also exists with a native C interface as *LAPACKE_dtrtrs*.

4.16.53 icmax1

icmax1 finds the index of the first vector element of maximum absolute value.

Based on ICAMAX from Level 1 BLAS. The change is to use the 'genuine' absolute value.

Syntax

Fortran specification:

```
use armpl_library

integer function icmax1(N, CX, INCX)
```

C specification:

```
#include "armpl.h"

armpl_int_t icmax1_(const armpl_int_t *n, const armpl_singlecomplex_t *cx,
                    const armpl_int_t *incx);
```

Parameters

N Input parameter.

N is INTEGER

The number of elements in the vector CX.

CX Input parameter.

CX is COMPLEX

CX is an array, dimension (N). The vector CX. The ICMAX1 function returns the index of its first element of maximum absolute value.

INCX Input parameter.

INCX is INTEGER

The spacing between successive values of CX. INCX >= 1.

Related Information

4.16.54 ilaclc

`ilaclc` scans A for its last non-zero column.

Syntax

Fortran specification:

```
use armpl_library

integer function ilaclc(M, N, A, LDA)
```

C specification:

```
#include "armpl.h"

armpl_int_t ilaclc_(const armpl_int_t *m, const armpl_int_t *n,
                    const armpl_singlecomplex_t *a, const armpl_int_t *lda);
```

Parameters

M Input parameter.

`M` is `INTEGER`

The number of rows of the matrix `A`.

N Input parameter.

`N` is `INTEGER`

The number of columns of the matrix `A`.

A Input parameter.

`A` is `COMPLEX`

`A` is an array, dimension (`LDA`, `N`). The m by n matrix `A`.

LDA Input parameter.

`LDA` is `INTEGER`

The leading dimension of the array `A`. `LDA` >= max(1, `M`).

Related Information

4.16.55 ilaclr

`ilaclr` scans A for its last non-zero row.

Syntax

Fortran specification:

```
use armpl_library

integer function ilaclr(M, N, A, LDA)
```

C specification:

```
#include "armpl.h"

armpl_int_t ilaclr_(const armpl_int_t *m, const armpl_int_t *n,
                    const armpl_singlecomplex_t *a, const armpl_int_t *lda);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A.

N Input parameter.

N is INTEGER

The number of columns of the matrix A.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The m by n matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

Related Information

4.16.56 iladlc

iladlc scans A for its last non-zero column.

Syntax

Fortran specification:

```
use armpl_library

integer function iladlc(M, N, A, LDA)
```

C specification:

```
#include "armpl.h"

armpl_int_t iladlc_(const armpl_int_t *m, const armpl_int_t *n,
                    const double *a, const armpl_int_t *lda);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A.

N Input parameter.

N is INTEGER

The number of columns of the matrix A.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). The m by n matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

Related Information

4.16.57 iladlr

iladlr scans A for its last non-zero row.

Syntax

Fortran specification:

```
use armpl_library

integer function iladlr(M, N, A, LDA)
```

C specification:

```
#include "armpl.h"

armpl_int_t iladlr_(const armpl_int_t *m, const armpl_int_t *n,
                    const double *a, const armpl_int_t *lda);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A.

N Input parameter.

N is INTEGER

The number of columns of the matrix A.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). The m by n matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

Related Information

4.16.58 ilaenv2stage

ILAENV2STAGE is called from the LAPACK routines to choose problem-dependent parameters for the local environment. See ISPEC for a description of the parameters. It sets problem and machine dependent parameters useful for *_2STAGE and related subroutines.

ILAENV2STAGE returns an INTEGER if ILAENV2STAGE >= 0: ILAENV2STAGE returns the value of the parameter if ILAENV2STAGE < 0: if ILAENV2STAGE = -k, the k-th argument had an This version provides a set of parameters which should give good, but not optimal, performance on many of the currently available computers for the 2-stage solvers. Users are encouraged to modify this subroutine to set the tuning parameters for their particular machine using the option and problem size information in the arguments.

This routine will not function correctly if it is converted to all lower case. Converting it to all upper case is allowed.

Syntax

Fortran specification:

```
use armpl_library

integer function ilaenv2stage(ISPEC, NAME, OPTS, N1, N2, N3, N4)
```

C specification:

```
#include "armpl.h"

armpl_int_t ilaenv2stage_(const armpl_int_t *ispec, const char *name,
                          const char *opts, const armpl_int_t *n1,
                          const armpl_int_t *n2, const armpl_int_t *n3,
                          const armpl_int_t *n4, ... );
```

Parameters

ISPEC Input parameter.

ISPEC is INTEGER

Specifies the parameter to be returned as the value of ILAENV2STAGE. = 1: the optimal blocksize nb for the reduction to BAND

= 2: the optimal blocksize ib for the eigenvectors singular vectors update routine

= 3: The length of the array that store the Housholder representation for the second stage Band to Tridiagonal or Bidiagonal

= 4: The workspace needed for the routine in input.

= 5: For future release.

NAME Input parameter.

NAME is CHARACTER*(*)

The name of the calling subroutine, in either upper case or lower case.

OPTS Input parameter.

OPTS is CHARACTER*(*)

The character options to the subroutine NAME, concatenated into a single character string. For example, UPLO = 'U', TRANS = 'T', and DIAG = 'N' for a triangular routine would be specified as OPTS = 'UTN'.

N1 Input parameter.

N1 is INTEGER

N2 Input parameter.

N2 is INTEGER

N3 Input parameter.

N3 is INTEGER

N4 Input parameter.

N4 is INTEGER

Problem dimensions for the subroutine NAME; these may not all be required.

Related Information

4.16.59 ilaslc

ilaslc scans A for its last non-zero column.

Syntax

Fortran specification:

```
use armpl_library

integer function ilaslc(M, N, A, LDA)
```

C specification:

```
#include "armpl.h"

armpl_int_t ilaslc_(const armpl_int_t *m, const armpl_int_t *n,
                    const float *a, const armpl_int_t *lda);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A.

N Input parameter.

N is INTEGER

The number of columns of the matrix A.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). The m by n matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

Related Information

4.16.60 ilaslr

`ilaslr` scans A for its last non-zero row.

Syntax

Fortran specification:

```
use armpl_library

integer function ilaslr(M, N, A, LDA)
```

C specification:

```
#include "armpl.h"

armpl_int_t ilaslr_(const armpl_int_t *m, const armpl_int_t *n,
                    const float *a, const armpl_int_t *lda);
```

Parameters

M Input parameter.

`M` is `INTEGER`

The number of rows of the matrix `A`.

N Input parameter.

`N` is `INTEGER`

The number of columns of the matrix `A`.

A Input parameter.

`A` is `REAL`

`A` is an array, dimension (`LDA`, `N`). The m by n matrix `A`.

LDA Input parameter.

`LDA` is `INTEGER`

The leading dimension of the array `A`. `LDA` >= max(1, `M`).

Related Information

4.16.61 ilazlc

`ilazlc` scans A for its last non-zero column.

Syntax

Fortran specification:

```
use armpl_library

integer function ilazlc(M, N, A, LDA)
```

C specification:

```
#include "armpl.h"

armpl_int_t ilazlc_(const armpl_int_t *m, const armpl_int_t *n,
                    const armpl_doublecomplex_t *a, const armpl_int_t *lda);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A.

N Input parameter.

N is INTEGER

The number of columns of the matrix A.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The m by n matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

Related Information

4.16.62 ilazlr

ilazlr scans A for its last non-zero row.

Syntax

Fortran specification:

```
use armpl_library

integer function ilazlr(M, N, A, LDA)
```

C specification:

```
#include "armpl.h"

armpl_int_t ilazlr_(const armpl_int_t *m, const armpl_int_t *n,
                    const armpl_doublecomplex_t *a, const armpl_int_t *lda);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A.

N Input parameter.

N is INTEGER

The number of columns of the matrix A.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The m by n matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

Related Information

4.16.63 izmax1

izmax1 finds the index of the first vector element of maximum absolute value.

Based on IZAMAX from Level 1 BLAS. The change is to use the 'genuine' absolute value.

Syntax

Fortran specification:

```
use armpl_library

integer function izmax1(N, ZX, INCX)
```

C specification:

```
#include "armpl.h"

armpl_int_t izmax1_(const armpl_int_t *n, const armpl_doublecomplex_t *zx,
                    const armpl_int_t *incx);
```

Parameters

N Input parameter.

N is INTEGER

The number of elements in the vector ZX.

ZX Input parameter.

ZX is COMPLEX*16

ZX is an array, dimension (N). The vector ZX. The IZMAX1 function returns the index of its first element of maximum absolute value.

INCX Input parameter.

INCX is INTEGER

The spacing between successive values of ZX. INCX >= 1.

Related Information

4.16.64 nancheck_flag

Syntax

Fortran specification:

```
use armpl_library

int nancheck_flag
```

C specification:

```
#include "armpl.h"

int nancheck_fla
```

Parameters

Related Information

4.16.65 scabs1

`scabs1` computes |Re(.)| + |Im(.)| of a complex number

Syntax

Fortran specification:

```
use armpl_library

real function scabs1(Z)
```

C specification:

```
#include "armpl.h"

float scabs1_(const armpl_singlecomplex_t *c);
```

Parameters

`Z` Input parameter.

`Z` is `COMPLEX`

Related Information

For this routine in other precisions, please see *dcabs1*.

4.16.66 sgetsls

`sgetsls` solves overdetermined or underdetermined real linear systems involving an M-by-N matrix A, using a tall skinny

The following options are provided:

* If TRANS = 'N' and m >= n: find the least squares solution of an overdetermined system, i.e., solve the least squares problem minimize || B - A*X ||.

* If TRANS = 'N' and m < n: find the minimum norm solution of an underdetermined system A * X = B.

* If TRANS = 'T' and m >= n: find the minimum norm solution of an undetermined system A^T * X = B.

* If TRANS = 'T' and m < n: find the least squares solution of an overdetermined system, i.e., solve the least squares problem minimize || B - A^T * X ||.

Several right hand side vectors b and solution vectors x can be handled in a single call; they are stored as the columns of the M-by-NRHS right hand side matrix B and the N-by-NRHS solution matrix X.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgetsls(TRANS, M, N, NRHS, A, LDA, B, LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgetsls_(const char *trans, const armpl_int_t *m, const armpl_int_t *n,
              const armpl_int_t *nrhs, float *a, const armpl_int_t *lda,
              float *b, const armpl_int_t *ldb, float *work,
              const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': the linear system involves A; = 'T': the linear system involves A^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >=0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, A is overwritten by details of its QR or LQ factorization as returned by SGEQR or SGELQ.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the matrix B of right hand side vectors, stored columnwise; B is M-by-NRHS if TRANS = 'N', or N-by-NRHS if TRANS = 'T'. On exit, if INFO = 0, B is overwritten by the solution vectors, stored columnwise: if TRANS = 'N' and m >= n, rows 1 to n of B contain the least squares solution vectors. if TRANS = 'N' and m < n, rows 1 to N of B contain the minimum norm solution vectors; if TRANS = 'T' and m >= n, rows 1 to M of B contain the minimum norm solution vectors; if TRANS = 'T' and m < n, rows 1 to M of B contain the least squares solution vectors.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= MAX(1, M, N).

WORK Output parameter.

(workspace) REAL

(workspace) is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) contains optimal (or either minimal or optimal, if query was assumed) LWORK. See LWORK for details.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If LWORK = -1 or -2, then a workspace query is assumed. If LWORK = -1, the routine calculates optimal size of WORK for the optimal performance and returns this value in WORK(1). If LWORK = -2, the routine calculates minimal size of WORK and returns this value in WORK(1).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element of the triangular factor of A is zero, so that A does not have full rank; the least squares solution could not be computed.

Related Information

For this routine in other precisions, please see *cgetsls*, *dgetsls* and *zgetsls*. It also exists with a native C interface as *LAPACKE_sgetsls*.

4.16.67 slasyf_rk

SLASYF_RK computes a partial factorization of a real symmetric matrix A using the bounded Bunch-Kaufman (rook) diagonal pivoting method. The partial factorization has the form:

A = (I U12) (A11 0) (I 0) if UPLO = 'U', or:

```
( 0  U22 ) (  0   D  ) ( U12**T U22**T )
```

A = (L11 0) (D 0) ($L11^T$ $L21^T$) if UPLO = 'L',

```
( L21  I ) (  0  A22 ) (  0       I    )
```

where the order of D is at most NB. The actual order is returned in the argument KB, and is either NB or NB-1, or N if N <= NB.

SLASYF_RK is an auxiliary routine called by SSYTRF_RK. It uses blocked code (calling Level 3 BLAS) to update the submatrix A11 (if UPLO = 'U') or A22 (if UPLO = 'L').

Syntax

Fortran specification:

```
use armpl_library

subroutine slasyf_rk(UPLO, N, NB, KB, A, LDA, E, IPIV, W, LDW, INFO)
```

C specification:

```
#include "armpl.h"

void slasyf_rk_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nb,
                armpl_int_t *kb, float *a, const armpl_int_t *lda, float *e,
                armpl_int_t *ipiv, float *w, const armpl_int_t *ldw,
                armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NB Input parameter.

NB is INTEGER

The maximum number of columns of the matrix A that should be factored. NB should be at least 2 to allow for 2-by-2 pivot blocks.

KB Output parameter.

KB is INTEGER

The number of columns of A that were actually factored. KB is either NB-1 or NB, or N if N <= NB.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U': the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced.

If UPLO = 'L': the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, contains: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Output parameter.

E is REAL

E is an array, dimension (N). On exit, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) is set to 0; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) is set to 0.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is set to 0 in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

IPIV describes the permutation matrix P in the factorization of matrix A as follows. The absolute value of IPIV(k) represents the index of row and column that were interchanged with the k-th row and column. The value of UPLO describes the order in which the interchanges were applied. Also, the sign of IPIV represents the block structure of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks which correspond to 1 or 2 interchanges at each factorization step.

If UPLO = 'U', (in factorization order, k decreases from N to 1): a) A single positive entry IPIV(k) > 0 means: D(k,k) is a 1-by-1 diagonal block. If IPIV(k) != k, rows and columns k and IPIV(k) were interchanged in the submatrix A(1:N,N-KB+1:N); If IPIV(k) = k, no interchange occurred.

b) A pair of consecutive negative entries IPIV(k) < 0 and IPIV(k-1) < 0 means: D(k-1:k,k-1:k) is a 2-by-2 diagonal block. (NOTE: negative entries in IPIV appear ONLY in pairs). 1) If -IPIV(k) != k, rows and columns k and -IPIV(k) were interchanged in the matrix A(1:N,N-KB+1:N). If -IPIV(k) = k, no interchange occurred. 2) If -IPIV(k-1) != k-1, rows and columns k-1 and -IPIV(k-1) were interchanged in the submatrix A(1:N,N-KB+1:N). If -IPIV(k-1) = k-1, no interchange occurred.

c) In both cases a) and b) is always ABS(IPIV(k)) <= k.

d) NOTE: Any entry IPIV(k) is always NONZERO on output.

If UPLO = 'L', (in factorization order, k increases from 1 to N): a) A single positive entry IPIV(k) > 0 means: D(k,k) is a 1-by-1 diagonal block. If IPIV(k) != k, rows and columns k and IPIV(k) were interchanged in the submatrix A(1:N,1:KB). If IPIV(k) = k, no interchange occurred.

b) A pair of consecutive negative entries IPIV(k) < 0 and IPIV(k+1) < 0 means: D(k:k+1,k:k+1) is a 2-by-2 diagonal block. (NOTE: negative entries in IPIV appear ONLY in pairs). 1) If -IPIV(k) != k, rows and columns k and -IPIV(k) were interchanged in the submatrix A(1:N,1:KB). If -IPIV(k) = k, no interchange occurred. 2) If -IPIV(k+1) != k+1, rows and columns k-1 and -IPIV(k-1) were interchanged in the submatrix A(1:N,1:KB). If -IPIV(k+1) = k+1, no interchange occurred.

c) In both cases a) and b) is always ABS(IPIV(k)) >= k.

d) NOTE: Any entry IPIV(k) is always NONZERO on output.

W Output parameter.

W is REAL

W is an array, dimension (LDW, NB) .

LDW Input parameter.

LDW is INTEGER

The leading dimension of the array W. LDW >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit

< 0: If INFO = -k, the k-th argument had an illegal value

> 0: If INFO = k, the matrix A is singular, because: If UPLO = 'U': column k in the upper triangular part of A contains all zeros. If UPLO = 'L': column k in the lower triangular part of A contains all zeros.

Therefore D(k,k) is exactly zero, and superdiagonal elements of column k of U (or subdiagonal elements of column k of L) are all zeros. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

NOTE: INFO only stores the first occurrence of a singularity, any subsequent occurrence of singularity is not stored in INFO even though the factorization always completes.

Related Information

For this routine in other precisions, please see *clasyf_rk*, *dlasyf_rk* and *zlasyf_rk*.

4.16.68 ssb2st_kernels

SSB2ST_KERNELS is an internal routine used by the SSYTRD_SB2ST subroutine.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssb2st_kernels(UPLO, WANTZ, TTYPE, ST, ED, SWEEP, N, NB, IB, A,
                          LDA, V, TAU, LDVT, WORK)
```

C specification:

```
#include "armpl.h"

void ssb2st_kernels_(const char *uplo, const armpl_int_t *wantz,
                     const armpl_int_t *ttype, const armpl_int_t *st,
                     const armpl_int_t *ed, const armpl_int_t *sweep,
                     const armpl_int_t *n, const armpl_int_t *nb,
                     const armpl_int_t *ib, float *a, const armpl_int_t *lda,
                     float *v, float *tau, const armpl_int_t *ldvt,
                     const float *work, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

WANTZ Input parameter.

WANTZ is LOGICAL which indicate if Eigenvalue are requested or both

Eigenvalue/Eigenvectors.

TTYPE Input parameter.

TTYPE is INTEGER

ST Input parameter.

ST is INTEGER

internal parameter for indices.

ED Input parameter.

ED is INTEGER

internal parameter for indices.

SWEEP Input parameter.

SWEEP is INTEGER

internal parameter for indices.

N Input parameter.

N is INTEGER. The order of the matrix A.

NB Input parameter.

NB is INTEGER. The size of the band.

IB Input parameter.

IB is INTEGER.

A Input and output parameter.

A is REAL array. A pointer to the matrix A.

LDA Input parameter.

LDA is INTEGER. The leading dimension of the matrix A.

V Output parameter.

V is REAL

V is an array, dimension 2*n if eigenvalues only are. requested or to be queried for vectors.

TAU Output parameter.

TAU is REAL

TAU is an array, dimension (2*n).. The scalar factors of the Householder reflectors are stored in this array.

LDVT Input parameter.

LDVT is INTEGER.

WORK Input parameter.

WORK is REAL array. Workspace of size nb.

n Input parameter.

The order of the matrix A.

Related Information

For this routine in other precisions, please see *dsb2st_kernels*.

4.16.69 ssbev_2stage

SSBEV_2STAGE computes all the eigenvalues and, optionally, eigenvectors of a real symmetric band matrix A using the 2stage technique for the reduction to tridiagonal.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssbev_2stage(JOBZ, UPLO, N, KD, AB, LDAB, W, Z, LDZ, WORK, LWORK,
                        INFO)
```

C specification:

```
#include "armpl.h"

void ssbev_2stage_(const char *jobz, const char *uplo, const armpl_int_t *n,
                   const armpl_int_t *kd, float *ab, const armpl_int_t *ldab,
                   float *w, float *z, const armpl_int_t *ldz, float *work,
                   const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors. Not available in this release.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is REAL

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, AB is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the first superdiagonal and the diagonal of the tridiagonal matrix T are returned in rows KD and KD+1 of AB, and if UPLO = 'L', the diagonal and first subdiagonal of T are returned in the first two rows of AB.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD + 1.

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is REAL

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the orthonormal eigenvectors of the matrix A, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension LWORK. On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= 1, when N <= 1; otherwise If JOBZ = 'N' and N > 1, LWORK must be queried. LWORK = MAX(1, dimension) where dimension = (2KD+1)* N + KD*NTHREADS + N where KD is the size of the band. NTHREADS is the number of threads used when openMP compilation is enabled, otherwise =1. If JOBZ = 'V' and N > 1, LWORK must be queried. Not yet available.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *dsbev_2stage*. It also exists with a native C interface as *LAPACKE_ssbev_2stage*.

4.16.70 ssbevd_2stage

SSBEVD_2STAGE computes all the eigenvalues and, optionally, eigenvectors of a real symmetric band matrix A using the 2stage technique for the reduction to tridiagonal. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssbevd_2stage(JOBZ, UPLO, N, KD, AB, LDAB, W, Z, LDZ, WORK, LWORK,
                         IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ssbevd_2stage_(const char *jobz, const char *uplo, const armpl_int_t *n,
                    const armpl_int_t *kd, float *ab, const armpl_int_t *ldab,
```

(continues on next page)

(continued from previous page)

```
                    float *w, float *z, const armpl_int_t *ldz, float *work,
                    const armpl_int_t *lwork, armpl_int_t *iwork,
                    const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors. Not available in this release.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is REAL

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, AB is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the first superdiagonal and the diagonal of the tridiagonal matrix T are returned in rows KD and KD+1 of AB, and if UPLO = 'L', the diagonal and first subdiagonal of T are returned in the first two rows of AB.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD + 1.

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is REAL

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the orthonormal eigenvectors of the matrix A, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension LWORK. On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= 1, when N <= 1; otherwise If JOBZ = 'N' and N > 1, LWORK must be queried. LWORK = MAX(1, dimension) where dimension = (2KD+1)* N + KD*NTHREADS + N where KD is the size of the band. NTHREADS is the number of threads used when openMP compilation is enabled, otherwise =1. If JOBZ = 'V' and N > 1, LWORK must be queried. Not yet available.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. If JOBZ = 'N' or N <= 1, LIWORK must be at least 1. If JOBZ = 'V' and N > 2, LIWORK must be at least 3 + 5*N.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *dsbevd_2stage*. It also exists with a native C interface as *LAPACKE_ssbevd_2stage*.

4.16.71 ssbevx_2stage

SSBEVX_2STAGE computes selected eigenvalues and, optionally, eigenvectors of a real symmetric band matrix A using the 2stage technique for the reduction to tridiagonal. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssbevx_2stage(JOBZ, RANGE, UPLO, N, KD, AB, LDAB, Q, LDQ, VL, VU,
                         IL, IU, ABSTOL, M, W, Z, LDZ, WORK, LWORK, IWORK,
                         IFAIL, INFO)
```

C specification:

```
#include "armpl.h"

void ssbevx_2stage_(const char *jobz, const char *range, const char *uplo,
                    const armpl_int_t *n, const armpl_int_t *kd, float *ab,
                    const armpl_int_t *ldab, float *q, const armpl_int_t *ldq,
                    const float *vl, const float *vu, const armpl_int_t *il,
                    const armpl_int_t *iu, const float *abstol,
                    armpl_int_t *m, float *w, float *z,
                    const armpl_int_t *ldz, float *work,
                    const armpl_int_t *lwork, armpl_int_t *iwork,
                    armpl_int_t *ifail, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors. Not available in this release.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found; = 'V': all eigenvalues in the half-open interval (VL,VU] will be found; = 'I': the IL-th through IU-th eigenvalues will be found.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is REAL

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, AB is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the first superdiagonal and the diagonal of the tridiagonal matrix T are returned in rows KD and KD+1 of AB, and if UPLO = 'L', the diagonal and first subdiagonal of T are returned in the first two rows of AB.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD + 1.

Q Output parameter.

Q is REAL

Q is an array, dimension (LDQ, N). If JOBZ = 'V', the N-by-N orthogonal matrix used in the reduction to tridiagonal form. If JOBZ = 'N', the array Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. If JOBZ = 'V', then LDQ >= max(1, N).

VL Input parameter.

VL is REAL

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is REAL

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is REAL

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing AB to tridiagonal form.

Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*SLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*SLAMCH('S').

See "Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy," by Demmel and Kahan, LAPACK Working Note #3.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is REAL

W is an array, dimension (N). The first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is REAL

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). If an eigenvector fails to converge, then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL. If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= 1, when N <= 1; otherwise If JOBZ = 'N' and N > 1, LWORK must be queried. LWORK = MAX(1, 7*N, dimension) where dimension = (2KD+1)* N + KD*NTHREADS + 2*N where KD is the size of the band. NTHREADS is the number of threads used when openMP compilation is enabled, otherwise =1. If JOBZ = 'V' and N > 1, LWORK must be queried. Not yet available

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (5*N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (N)

If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, then i eigenvectors failed to converge. Their indices are stored in array IFAIL.

Related Information

For this routine in other precisions, please see *dsbevx_2stage*. It also exists with a native C interface as *LAPACKE_ssbevx_2stage*.

4.16.72 ssyconvf

If parameter WAY = 'C': SSYCONVF converts the factorization output format used in SSYTRF provided on entry in parameter A into the factorization output format used in SSYTRF_RK (or SSYTRF_BK) that is stored on exit in parameters A and E. It also coverts in place details of the intechanges stored in IPIV from the format used in SSYTRF into the format used in SSYTRF_RK (or SSYTRF_BK).

If parameter WAY = 'R': SSYCONVF performs the conversion in reverse direction, i.e. converts the factorization output format used in SSYTRF_RK (or SSYTRF_BK) provided on entry in parameters A and E into the factorization output format used in SSYTRF that is stored on exit in parameter A. It also coverts in place details of the intechanges stored in IPIV from the format used in SSYTRF_RK (or SSYTRF_BK) into the format used in SSYTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssyconvf(UPLO, WAY, N, A, LDA, E, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void ssyconvf_(const char *uplo, const char *way, const armpl_int_t *n,
               float *a, const armpl_int_t *lda, float *e, armpl_int_t *ipiv,
               armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix A. = 'U': Upper triangular = 'L': Lower triangular

WAY Input parameter.

WAY is CHARACTER*1

= 'C': Convert = 'R': Revert

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). 1) If WAY ='C':

On entry, contains factorization details in format used in SSYTRF: a) all elements of the symmetric block diagonal matrix D on the diagonal of A and on superdiagonal (or subdiagonal) of A, and b) If UPLO = 'U': multipliers used to obtain factor U in the superdiagonal part of A. If UPLO = 'L': multipliers used to obtain factor L in the superdiagonal part of A.

On exit, contains factorization details in format used in SSYTRF_RK or SSYTRF_BK: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

2) If WAY = 'R':

On entry, contains factorization details in format used in SSYTRF_RK or SSYTRF_BK: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

On exit, contains factorization details in format used in SSYTRF: a) all elements of the symmetric block diagonal matrix D on the diagonal of A and on superdiagonal (or subdiagonal) of A, and b) If UPLO = 'U': multipliers used to obtain factor U in the superdiagonal part of A. If UPLO = 'L': multipliers used to obtain factor L in the superdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Input and output parameter.

E is REAL

E is an array, dimension (N). 1) If WAY ='C':

On entry, just a workspace.

On exit, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) is set to 0; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) is set to 0.

2) If WAY = 'R':

On entry, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i),i=2:N, E(1) not referenced; If UPLO = 'L': E(i) = D(i+1,i),i=1:N-1, E(N) not referenced.

On exit, is not changed

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

1) If WAY ='C': On entry, details of the interchanges and the block structure of D in the format used in SSYTRF. On exit, details of the interchanges and the block structure of D in the format used in SSYTRF_RK (or SSYTRF_BK).

1) If WAY ='R': On entry, details of the interchanges and the block structure of D in the format used in SSYTRF_RK (or SSYTRF_BK). On exit, details of the interchanges and the block structure of D in the format used in SSYTRF.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csyconvf*, *dsyconvf* and *zsyconvf*.

4.16.73 ssyconvf_rook

If parameter WAY = 'C': SSYCONVF_ROOK converts the factorization output format used in SSYTRF_ROOK provided on entry in parameter A into the factorization output format used in SSYTRF_RK (or SSYTRF_BK) that is stored on exit in parameters A and E. IPIV format for SSYTRF_ROOK and SSYTRF_RK (or SSYTRF_BK) is the same and is not converted.

If parameter WAY = 'R': SSYCONVF_ROOK performs the conversion in reverse direction, i.e. converts the factorization output format used in SSYTRF_RK (or SSYTRF_BK) provided on entry in parameters A and E into the factorization output format used in SSYTRF_ROOK that is stored on exit in parameter A. IPIV format for SSYTRF_ROOK and SSYTRF_RK (or SSYTRF_BK) is the same and is not converted.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssyconvf_rook(UPLO, WAY, N, A, LDA, E, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void ssyconvf_rook_(const char *uplo, const char *way, const armpl_int_t *n,
                    float *a, const armpl_int_t *lda, float *e,
                    const armpl_int_t *ipiv, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix A. = 'U': Upper triangular = 'L': Lower triangular

WAY Input parameter.

WAY is CHARACTER*1

= 'C': Convert = 'R': Revert

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). 1) If WAY ='C':

On entry, contains factorization details in format used in SSYTRF_ROOK: a) all elements of the symmetric block diagonal matrix D on the diagonal of A and on superdiagonal (or subdiagonal) of A, and b) If UPLO = 'U': multipliers used to obtain factor U in the superdiagonal part of A. If UPLO = 'L': multipliers used to obtain factor L in the superdiagonal part of A.

On exit, contains factorization details in format used in SSYTRF_RK or SSYTRF_BK: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

2) If WAY = 'R':

On entry, contains factorization details in format used in SSYTRF_RK or SSYTRF_BK: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

On exit, contains factorization details in format used in SSYTRF_ROOK: a) all elements of the symmetric block diagonal matrix D on the diagonal of A and on superdiagonal (or subdiagonal) of A, and b) If UPLO = 'U': multipliers used to obtain factor U in the superdiagonal part of A. If UPLO = 'L': multipliers used to obtain factor L in the superdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Input and output parameter.

E is REAL

E is an array, dimension (N). 1) If WAY ='C':

On entry, just a workspace.

On exit, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) is set to 0; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) is set to 0.

2) If WAY = 'R':

On entry, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i),i=2:N, E(1) not referenced; If UPLO = 'L': E(i) = D(i+1,i),i=1:N-1, E(N) not referenced.

On exit, is not changed

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

On entry, details of the interchanges and the block structure of D as determined: 1) by SSYTRF_ROOK, if WAY ='C'; 2) by SSYTRF_RK (or SSYTRF_BK), if WAY ='R'. The IPIV format is the same for all these routines.

On exit, is not changed.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csyconvf_rook*, *dsyconvf_rook* and *zsyconvf_rook*.

4.16.74 ssyev_2stage

SSYEV_2STAGE computes all eigenvalues and, optionally, eigenvectors of a real symmetric matrix A using the 2stage technique for the reduction to tridiagonal.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssyev_2stage(JOBZ, UPLO, N, A, LDA, W, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ssyev_2stage_(const char *jobz, const char *uplo, const armpl_int_t *n,
                   float *a, const armpl_int_t *lda, float *w, float *work,
                   const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors. Not available in this release.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A. On exit, if JOBZ = 'V', then if INFO = 0, A contains the orthonormal eigenvectors of the matrix A. If JOBZ = 'N', then on exit the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension LWORK. On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= 1, when N <= 1; otherwise If JOBZ = 'N' and N > 1, LWORK must be queried. LWORK = MAX(1, dimension) where dimension = max(stage1,stage2) + (KD+1)* N + 2*N = N*KD + N*max(KD+1,FACTOPTNB) + max(2*KD*KD, KD*NTHREADS) + (KD+1)* N + 2*N where KD is the blocking size of the reduction, FACTOPTNB is the blocking used by the QR or LQ algorithm, usually FACTOPTNB=128 is a good choice NTHREADS is the number of threads used when openMP compilation is enabled, otherwise =1. If JOBZ = 'V' and N > 1, LWORK must be queried. Not yet available

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *dsyev_2stage*. It also exists with a native C interface as *LAPACKE_ssyev_2stage*.

4.16.75 ssyevd_2stage

SSYEVD_2STAGE computes all eigenvalues and, optionally, eigenvectors of a real symmetric matrix A using the 2stage technique for the reduction to tridiagonal. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssyevd_2stage(JOBZ, UPLO, N, A, LDA, W, WORK, LWORK, IWORK, LIWORK,
                         INFO)
```

C specification:

```
#include "armpl.h"

void ssyevd_2stage_(const char *jobz, const char *uplo, const armpl_int_t *n,
                    float *a, const armpl_int_t *lda, float *w, float *work,
                    const armpl_int_t *lwork, armpl_int_t *iwork,
                    const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors. Not available in this release.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A. On exit, if JOBZ = 'V', then if INFO = 0, A contains the orthonormal eigenvectors of the matrix A. If JOBZ = 'N', then on exit the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

WORK Output parameter.

WORK is REAL

WORK is an array,. dimension (LWORK) On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If N <= 1, LWORK must be at least 1. If JOBZ = 'N' and N > 1, LWORK must be queried. LWORK = MAX(1, dimension) where dimension = max(stage1,stage2) + (KD+1)* N + 2*N+1 = N*KD + N*max(KD+1,FACTOPTNB) + max(2*KD*KD, KD*NTHREADS) + (KD+1)* N + 2*N+1 where KD is the blocking size of the reduction, FACTOPTNB is the blocking used by the QR or LQ algorithm, usually FACTOPTNB=128 is a good choice NTHREADS is the number of threads used when openMP compilation is enabled, otherwise =1. If JOBZ = 'V' and N > 1, LWORK must be at least 1 + 6* N + 2*N**2.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. If N <= 1, LIWORK must be at least 1. If JOBZ = 'N' and N > 1, LIWORK must be at least 1. If JOBZ = 'V' and N > 1, LIWORK must be at least 3 + 5*N.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK and IWORK arrays, returns these values as the first entries of the WORK and IWORK arrays, and no error message related to LWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i and JOBZ = 'N', then the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero; if INFO = i and JOBZ = 'V', then the algorithm failed to compute an eigenvalue while working on the submatrix lying in rows and columns INFO/(N+1) through mod(INFO,N+1).

Related Information

For this routine in other precisions, please see *dsyevd_2stage*. It also exists with a native C interface as *LAPACKE_ssyevd_2stage*.

4.16.76 ssyevr_2stage

SSYEVR_2STAGE computes selected eigenvalues and, optionally, eigenvectors of a real symmetric matrix A using the 2stage technique for the reduction to tridiagonal. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

SSYEVR_2STAGE first reduces the matrix A to tridiagonal form T with a call to SSYTRD. Then, whenever possible, SSYEVR_2STAGE calls SSTEMR to compute the eigenspectrum using Relatively Robust Representations. SSTEMR computes eigenvalues by the dqds algorithm, while orthogonal eigenvectors are computed from various "good" L D L^T representations (also known as Relatively Robust Representations). Gram-Schmidt orthogonalization is avoided as far as possible. More specifically, the various steps of the algorithm are as follows.

For each unreduced block (submatrix) of T,

```
(a) Compute T - sigma I  = L D L^T, so that L and D
    define all the wanted eigenvalues to high relative accuracy.
    This means that small relative changes in the entries of D and L
    cause only small relative changes in the eigenvalues and
    eigenvectors. The standard (unfactored) representation of the
    tridiagonal matrix T does not have this property in general.
(b) Compute the eigenvalues to suitable accuracy.
    If the eigenvectors are desired, the algorithm attains full
    accuracy of the computed eigenvalues only right before
    the corresponding vectors have to be computed, see steps c) and d).
(c) For each cluster of close eigenvalues, select a new
    shift close to the cluster, find a new factorization, and refine
    the shifted eigenvalues to suitable accuracy.
(d) For each eigenvalue with a large enough relative separation compute
    the corresponding eigenvector by forming a rank revealing twisted
    factorization. Go back to (c) for any clusters that remain.
```

The desired accuracy of the output can be specified by the input parameter ABSTOL.

For more details, see SSTEMR's documentation and: - Inderjit S. Dhillon and Beresford N. Parlett: "Multiple representations

```
 to compute orthogonal eigenvectors of symmetric tridiagonal matrices,"
Linear Algebra and its Applications, 387(1), pp. 1-28, August 2004.
```

- Inderjit Dhillon and Beresford Parlett: "Orthogonal Eigenvectors and

```
 Relative Gaps," SIAM Journal on Matrix Analysis and Applications, Vol. 25,
2004.  Also LAPACK Working Note 154.
```

- Inderjit Dhillon: "A new O(n^2) algorithm for the symmetric

```
 tridiagonal eigenvalue/eigenvector problem",
Computer Science Division Technical Report No. UCB/CSD-97-971,
UC Berkeley, May 1997.
```

Note 1 : SSYEVR_2STAGE calls SSTEMR when the full spectrum is requested on machines which conform to the ieee-754 floating point standard. SSYEVR_2STAGE calls SSTEBZ and SSTEIN on non-ieee machines and when partial spectrum requests are made.

Normal execution of SSTEMR may create NaNs and infinities and hence may abort due to a floating point exception in environments which do not handle NaNs and infinities in the ieee standard default manner.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssyevr_2stage(JOBZ, RANGE, UPLO, N, A, LDA, VL, VU, IL, IU, ABSTOL,
                         M, W, Z, LDZ, ISUPPZ, WORK, LWORK, IWORK, LIWORK,
                         INFO)
```

C specification:

```
#include "armpl.h"

void ssyevr_2stage_(const char *jobz, const char *range, const char *uplo,
                    const armpl_int_t *n, float *a, const armpl_int_t *lda,
                    const float *vl, const float *vu, const armpl_int_t *il,
                    const armpl_int_t *iu, const float *abstol,
                    armpl_int_t *m, float *w, float *z,
                    const armpl_int_t *ldz, armpl_int_t *isuppz, float *work,
                    const armpl_int_t *lwork, armpl_int_t *iwork,
                    const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors. Not available in this release.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found. = 'V': all eigenvalues in the half-open interval (VL,VU] will be found. = 'I': the IL-th through IU-th eigenvalues will be found. For RANGE = 'V' or 'I' and IU - IL < N - 1, SSTEBZ and SSTEIN are called

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A. On exit, the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

VL Input parameter.

VL is REAL

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is REAL

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is REAL

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing A to tridiagonal form.

See "Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy," by Demmel and Kahan, LAPACK Working Note #3.

If high relative accuracy is important, set ABSTOL to SLAMCH('Safe minimum'). Doing so will guarantee that eigenvalues are computed to high relative accuracy when possible in future releases. The current code does not make any guarantees about high relative accuracy, but future releases will. See J. Barlow and J. Demmel, "Computing Accurate Eigensystems of Scaled Diagonally Dominant Matrices", LAPACK Working Note #7, for a discussion of which matrices define their eigenvalues to high relative accuracy.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is REAL

W is an array, dimension (N). The first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is REAL

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used. Supplying N columns is always safe.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

ISUPPZ Output parameter.

ISUPPZ is INTEGER array, dimension (2*max(1, M))

The support of the eigenvectors in Z, i.e., the indices indicating the nonzero elements in Z. The i-th eigenvector is nonzero only in elements ISUPPZ(2*i-1) through ISUPPZ(2*i). This is an output of SSTEMR (tridiagonal matrix). The support of the eigenvectors of A is typically 1:N because of the orthogonal transformations applied by SORMTR. Implemented only for RANGE = 'A' or 'I' and IU - IL = N - 1

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If JOBZ = 'N' and N > 1, LWORK must be queried. LWORK = MAX(1, 26*N, dimension) where dimension = max(stage1,stage2) + (KD+1)* N + 5*N = N*KD + N*max(KD+1,FACTOPTNB) + max(2*KD*KD, KD*NTHREADS) + (KD+1)* N + 5*N where KD is the blocking size of the reduction, FACTOPTNB is the blocking used by the QR or LQ algorithm, usually FACTOPTNB=128 is a good choice NTHREADS is the number of threads used when openMP compilation is enabled, otherwise =1. If JOBZ = 'V' and N > 1, LWORK must be queried. Not yet available

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. LIWORK >= max(1,10*N).

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the IWORK array, returns this value as the first entry of the IWORK array, and no error message related to LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: Internal error

Related Information

For this routine in other precisions, please see *dsyevr_2stage*. It also exists with a native C interface as *LAPACKE_ssyevr_2stage*.

4.16.77 ssyevx_2stage

SSYEVX_2STAGE computes selected eigenvalues and, optionally, eigenvectors of a real symmetric matrix A using the 2stage technique for the reduction to tridiagonal. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssyevx_2stage(JOBZ, RANGE, UPLO, N, A, LDA, VL, VU, IL, IU, ABSTOL,
                         M, W, Z, LDZ, WORK, LWORK, IWORK, IFAIL, INFO)
```

C specification:

```
#include "armpl.h"

void ssyevx_2stage_(const char *jobz, const char *range, const char *uplo,
                    const armpl_int_t *n, float *a, const armpl_int_t *lda,
                    const float *vl, const float *vu, const armpl_int_t *il,
                    const armpl_int_t *iu, const float *abstol,
                    armpl_int_t *m, float *w, float *z,
                    const armpl_int_t *ldz, float *work,
                    const armpl_int_t *lwork, armpl_int_t *iwork,
                    armpl_int_t *ifail, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors. Not available in this release.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found. = 'V': all eigenvalues in the half-open interval (VL,VU] will be found. = 'I': the IL-th through IU-th eigenvalues will be found.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A. On exit, the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

VL Input parameter.

VL is REAL

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is REAL

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is REAL

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing A to tridiagonal form.

Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*SLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*SLAMCH('S').

See "Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy," by Demmel and Kahan, LAPACK Working Note #3.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is REAL

W is an array, dimension (N). On normal exit, the first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is REAL

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column

of Z holding the eigenvector associated with W(i). If an eigenvector fails to converge, then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL. If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= 1, when N <= 1; otherwise If JOBZ = 'N' and N > 1, LWORK must be queried. LWORK = MAX(1, 8*N, dimension) where dimension = max(stage1,stage2) + (KD+1)* N + 3*N = N*KD + N*max(KD+1,FACTOPTNB) + max(2*KD*KD, KD*NTHREADS) + (KD+1)* N + 3*N where KD is the blocking size of the reduction, FACTOPTNB is the blocking used by the QR or LQ algorithm, usually FACTOPTNB=128 is a good choice NTHREADS is the number of threads used when openMP compilation is enabled, otherwise =1. If JOBZ = 'V' and N > 1, LWORK must be queried. Not yet available

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (5*N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (N)

If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, then i eigenvectors failed to converge. Their indices are stored in array IFAIL.

Related Information

For this routine in other precisions, please see *dsyevx_2stage*. It also exists with a native C interface as *LAPACKE_ssyevx_2stage*.

4.16.78 ssygv_2stage

SSYGV_2STAGE computes all the eigenvalues, and optionally, the eigenvectors of a real generalized symmetric-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x. Here A and B are assumed to be symmetric and B is also positive definite. This routine use the 2stage technique for the reduction to tridiagonal which showed higher performance on recent architecture and for large sizes N>2000.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssygv_2stage(ITYPE, JOBZ, UPLO, N, A, LDA, B, LDB, W, WORK, LWORK,
                        INFO)
```

C specification:

```
#include "armpl.h"

void ssygv_2stage_(const armpl_int_t *itype, const char *jobz,
                   const char *uplo, const armpl_int_t *n, float *a,
                   const armpl_int_t *lda, float *b, const armpl_int_t *ldb,
                   float *w, float *work, const armpl_int_t *lwork,
                   armpl_int_t *info, ... );
```

Parameters

ITYPE Input parameter.

ITYPE is INTEGER

Specifies the problem type to be solved: = 1: A*x = (lambda)*B*x = 2: A*B*x = (lambda)*x = 3: B*A*x = (lambda)*x

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors. Not available in this release.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangles of A and B are stored; = 'L': Lower triangles of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A.

On exit, if JOBZ = 'V', then if INFO = 0, A contains the matrix Z of eigenvectors. The eigenvectors are normalized as follows: if ITYPE = 1 or 2, Z^T *B*Z = I; if ITYPE = 3, Z^T *inv(B)*Z = I. If JOBZ = 'N', then on exit the upper triangle (if UPLO='U') or the lower triangle (if UPLO='L') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, N). On entry, the symmetric positive definite matrix B. If UPLO = 'U', the leading N-by-N upper triangular part of B contains the upper triangular part of the matrix B. If UPLO = 'L', the leading N-by-N lower triangular part of B contains the lower triangular part of the matrix B.

On exit, if INFO <= N, the part of B containing the matrix is overwritten by the triangular factor U or L from the Cholesky factorization B = U^T *U or B = L*L^T .

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

W Output parameter.

W is REAL

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= 1, when N <= 1; otherwise If JOBZ = 'N' and N > 1, LWORK must be queried. LWORK = MAX(1, dimension) where dimension = max(stage1,stage2) + (KD+1)* N + 2*N = N*KD + N*max(KD+1,FACTOPTNB) + max(2*KD*KD, KD*NTHREADS) + (KD+1)* N + 2*N where KD is the blocking size of the reduction, FACTOPTNB is the blocking used by the QR or LQ algorithm, usually FACTOPTNB=128 is a good choice NTHREADS is the number of threads used when openMP compilation is enabled, otherwise =1. If JOBZ = 'V' and N > 1, LWORK must be queried. Not yet available

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: SPOTRF or SSYEV returned an error code: <= N: if INFO = i, SSYEV failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero; > N: if INFO = N + i, for 1 <= i <= N, then the leading minor of order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *dsygv_2stage*. It also exists with a native C interface as *LAPACKE_ssygv_2stage*.

4.16.79 ssysv_aa_2stage

SSYSV_AA_2STAGE computes the solution to a real system of linear equations

```
A * X = B,
```

where A is an N-by-N symmetric matrix and X and B are N-by-NRHS matrices.

Aasen's 2-stage algorithm is used to factor A as

```
A = U * T * U**T,  if UPLO = 'U', or
A = L * T * L**T,  if UPLO = 'L',
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and T is symmetric and band. The matrix T is then LU-factored with partial pivoting. The factored form of A is then used to solve the system of equations A * X = B.

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssysv_aa_2stage(UPLO, N, NRHS, A, LDA, TB, LTB, IPIV, IPIV2, B,
                           LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ssysv_aa_2stage_(const char *uplo, const armpl_int_t *n,
                      const armpl_int_t *nrhs, float *a,
                      const armpl_int_t *lda, float *tb,
                      const armpl_int_t *ltb, armpl_int_t *ipiv,
                      armpl_int_t *ipiv2, float *b, const armpl_int_t *ldb,
                      float *work, const armpl_int_t *lwork,
                      armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, L is stored below (or above) the subdiaonal blocks, when UPLO is 'L' (or 'U').

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TB Output parameter.

TB is REAL

TB is an array, dimension (LTB). On exit, details of the LU factorization of the band matrix.

LTB Input parameter.

The size of the array TB. LTB >= 4*N, internally

used to select NB such that LTB >= (3*NB+1)*N.

If LTB = -1, then a workspace query is assumed; the routine only calculates the optimal size of LTB, returns this value as the first entry of TB, and no error message related to LTB is issued by XERBLA.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

On exit, it contains the details of the interchanges, i.e., the row and column k of A were interchanged with the row and column IPIV(k).

IPIV2 Output parameter.

IPIV is INTEGER array, dimension (N)

On exit, it contains the details of the interchanges, i.e., the row and column k of T were interchanged with the row and column IPIV(k).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Output parameter.

WORK is REAL workspace of size LWORK

LWORK Input parameter.

The size of WORK. LWORK >= N, internally used to select NB

such that LWORK >= N*NB.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, band LU factorization failed on i-th column

Related Information

For this routine in other precisions, please see *csysv_aa_2stage*, *dsysv_aa_2stage* and *zsysv_aa_2stage*. It also exists with a native C interface as *LAPACKE_ssysv_aa_2stage*.

4.16.80 ssytf2_rk

SSYTF2_RK computes the factorization of a real symmetric matrix A using the bounded Bunch-Kaufman (rook) diagonal pivoting method:

```
A = P*U*D*(U**T)*(P**T) or A = P*L*D*(L**T)*(P**T),
```

where U (or L) is unit upper (or lower) triangular matrix, U^T (or L^T) is the transpose of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the unblocked version of the algorithm, calling Level 2 BLAS. For more information see Further Details section.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssytf2_rk(UPLO, N, A, LDA, E, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void ssytf2_rk_(const char *uplo, const armpl_int_t *n, float *a,
                const armpl_int_t *lda, float *e, armpl_int_t *ipiv,
                armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U': the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced.

If UPLO = 'L': the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, contains: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Output parameter.

E is REAL

E is an array, dimension (N). On exit, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) is set to 0; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) is set to 0.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is set to 0 in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

IPIV describes the permutation matrix P in the factorization of matrix A as follows. The absolute value of IPIV(k) represents the index of row and column that were interchanged with the k-th row and column. The value of UPLO describes the order in which the interchanges were applied. Also, the sign of IPIV represents the block structure of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks which correspond to 1 or 2 interchanges at each factorization step. For more info see Further Details section.

If UPLO = 'U', (in factorization order, k decreases from N to 1): a) A single positive entry IPIV(k) > 0 means: D(k,k) is a 1-by-1 diagonal block. If IPIV(k) != k, rows and columns k and IPIV(k) were interchanged in the matrix A(1:N,1:N); If IPIV(k) = k, no interchange occurred.

b) A pair of consecutive negative entries IPIV(k) < 0 and IPIV(k-1) < 0 means: D(k-1:k,k-1:k) is a 2-by-2 diagonal block. (NOTE: negative entries in IPIV appear ONLY in pairs). 1) If -IPIV(k) != k, rows and columns k and -IPIV(k) were interchanged in the matrix A(1:N,1:N). If -IPIV(k) = k, no interchange occurred. 2) If -IPIV(k-1) != k-1, rows and columns k-1 and -IPIV(k-1) were interchanged in the matrix A(1:N,1:N). If -IPIV(k-1) = k-1, no interchange occurred.

c) In both cases a) and b), always ABS(IPIV(k)) <= k.

d) NOTE: Any entry IPIV(k) is always NONZERO on output.

If UPLO = 'L', (in factorization order, k increases from 1 to N): a) A single positive entry IPIV(k) > 0 means: D(k,k) is a 1-by-1 diagonal block. If IPIV(k) != k, rows and columns k and IPIV(k) were interchanged in the matrix A(1:N,1:N). If IPIV(k) = k, no interchange occurred.

b) A pair of consecutive negative entries IPIV(k) < 0 and IPIV(k+1) < 0 means: D(k:k+1,k:k+1) is a 2-by-2 diagonal block. (NOTE: negative entries in IPIV appear ONLY in pairs). 1) If -IPIV(k) != k, rows and columns k and -IPIV(k) were interchanged in the matrix A(1:N,1:N). If -IPIV(k) = k, no interchange occurred. 2) If -IPIV(k+1) != k+1, rows and columns k-1 and -IPIV(k-1) were interchanged in the matrix A(1:N,1:N). If -IPIV(k+1) = k+1, no interchange occurred.

c) In both cases a) and b), always ABS(IPIV(k)) >= k.

d) NOTE: Any entry IPIV(k) is always NONZERO on output.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit

< 0: If INFO = -k, the k-th argument had an illegal value

> 0: If INFO = k, the matrix A is singular, because: If UPLO = 'U': column k in the upper triangular part of A contains all zeros. If UPLO = 'L': column k in the lower triangular part of A contains all zeros.

Therefore D(k,k) is exactly zero, and superdiagonal elements of column k of U (or subdiagonal elements of column k of L) are all zeros. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

NOTE: INFO only stores the first occurrence of a singularity, any subsequent occurrence of singularity is not stored in INFO even though the factorization always completes.

Related Information

For this routine in other precisions, please see *csytf2_rk*, *dsytf2_rk* and *zsytf2_rk*.

4.16.81 ssytrd_2stage

SSYTRD_2STAGE reduces a real symmetric matrix A to real symmetric tridiagonal form T by a orthogonal similarity transformation: $Q1^T$ $Q2^T$ * A * Q2 * Q1 = T.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssytrd_2stage(VECT, UPLO, N, A, LDA, D, E, TAU, HOUS2, LHOUS2,
                         WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ssytrd_2stage_(const char *vect, const char *uplo, const armpl_int_t *n,
                    float *a, const armpl_int_t *lda, float *d, float *e,
                    float *tau, float *hous2, const armpl_int_t *lhous2,
                    float *work, const armpl_int_t *lwork, armpl_int_t *info,
                    ... );
```

Parameters

VECT Input parameter.

VECT is CHARACTER*1

= 'N': No need for the Housholder representation, in particular for the second stage (Band to tridiagonal) and thus LHOUS2 is of size max(1, 4*N); = 'V': the Householder representation is needed to either generate Q1 Q2 or to apply Q1 Q2, then LHOUS2 is to be queried and computed. (NOT AVAILABLE IN THIS RELEASE).

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, if UPLO = 'U', the band superdiagonal of A are overwritten by the corresponding elements of the internal band-diagonal matrix AB, and the elements above the KD superdiagonal, with the array TAU, represent the orthogonal matrix Q1 as a product of elementary reflectors; if UPLO = 'L', the diagonal and band subdiagonal of A are

over- written by the corresponding elements of the internal band-diagonal matrix AB, and the elements below the KD subdiagonal, with the array TAU, represent the orthogonal matrix Q1 as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

D Output parameter.

D is REAL

D is an array, dimension (N). The diagonal elements of the tridiagonal matrix T.

E Output parameter.

E is REAL

E is an array, dimension (N-1). The off-diagonal elements of the tridiagonal matrix T.

TAU Output parameter.

TAU is REAL

TAU is an array, dimension (N-KD). The scalar factors of the elementary reflectors of the first stage (see Further Details).

HOUS2 Output parameter.

HOUS2 is REAL

HOUS2 is an array, dimension LHOUS2, that. store the Householder representation of the stage2 band to tridiagonal.

LHOUS2 Input parameter.

LHOUS2 is INTEGER

The dimension of the array HOUS2. LHOUS2 = MAX(1, dimension) If LWORK = -1, or LHOUS2=-1, then a query is assumed; the routine only calculates the optimal size of the HOUS2 array, returns this value as the first entry of the HOUS2 array, and no error message related to LHOUS2 is issued by XERBLA. LHOUS2 = MAX(1, dimension) where dimension = 4* N if VECT='N' not available now if VECT='H'

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK = MAX(1, dimension) If LWORK = -1, or LHOUS2=-1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA. LWORK = MAX(1, dimension) where dimension = max(stage1,stage2) + (KD+1)*N = N*KD + N*max(KD+1,FACTOPTNB) + max(2*KD*KD, KD*NTHREADS) + (KD+1)* N where KD is the blocking size of the reduction, FACTOPTNB is the blocking used by the QR or LQ algorithm, usually FACTOPTNB=128 is a good choice NTHREADS is the number of threads used when openMP compilation is enabled, otherwise =1.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dsytrd_2stage*.

4.16.82 ssytrd_sy2sb

SSYTRD_SY2SB reduces a real symmetric matrix A to real symmetric band-diagonal form AB by a orthogonal similarity transformation: Q^T * A * Q = AB.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssytrd_sy2sb(UPLO, N, KD, A, LDA, AB, LDAB, TAU, WORK, LWORK,
                        INFO)
```

C specification:

```
#include "armpl.h"

void ssytrd_sy2sb_(const char *uplo, const armpl_int_t *n,
                   const armpl_int_t *kd, float *a, const armpl_int_t *lda,
                   float *ab, const armpl_int_t *ldab, float *tau,
                   float *work, const armpl_int_t *lwork, armpl_int_t *info,
                   ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the reduced matrix if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0. The reduced matrix is stored in the array AB.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, if UPLO = 'U', the diagonal and first superdiagonal of A are overwritten by the corresponding elements of the tridiagonal matrix T, and the elements above the first superdiagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors; if UPLO = 'L', the diagonal and first subdiagonal of A are over- written by the corresponding elements of the tridiagonal matrix T, and the elements below the

first subdiagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AB Output parameter.

AB is REAL

AB is an array, dimension (LDAB, N). On exit, the upper or lower triangle of the symmetric band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

TAU Output parameter.

TAU is REAL

TAU is an array, dimension (N-KD). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (LWORK). On exit, if INFO = 0, or if LWORK=-1, WORK(1) returns the size of LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK which should be calculated by a workspace query. LWORK = MAX(1, LWORK_QUERY) If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA. LWORK_QUERY = N* KD + N*max(KD,FACTOPTNB) + 2*KD*KD where FACTOPTNB is the blocking used by the QR or LQ algorithm, usually FACTOPTNB=128 is a good choice otherwise putting LWORK=-1 will provide the size of WORK.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dsytrd_sy2sb*.

4.16.83 ssytrf_aa_2stage

SSYTRF_AA_2STAGE computes the factorization of a real symmetric matrix A using the Aasen's algorithm. The form of the factorization is

```
A = U*T*U**T  or  A = L*T*L**T
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and T is a symmetric band matrix with the bandwidth of NB (NB is internally selected and stored in TB(1), and T is LU factorized with partial pivoting).

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssytrf_aa_2stage(UPLO, N, A, LDA, TB, LTB, IPIV, IPIV2, WORK,
                            LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ssytrf_aa_2stage_(const char *uplo, const armpl_int_t *n, float *a,
                       const armpl_int_t *lda, float *tb,
                       const armpl_int_t *ltb, armpl_int_t *ipiv,
                       armpl_int_t *ipiv2, float *work,
                       const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, L is stored below (or above) the subdiaonal blocks, when UPLO is 'L' (or 'U').

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TB Output parameter.

TB is REAL

TB is an array, dimension (LTB). On exit, details of the LU factorization of the band matrix.

LTB Input parameter.

The size of the array TB. LTB >= 4*N, internally

used to select NB such that LTB >= (3*NB+1)*N.

If LTB = -1, then a workspace query is assumed; the routine only calculates the optimal size of LTB, returns this value as the first entry of TB, and no error message related to LTB is issued by XERBLA.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

On exit, it contains the details of the interchanges, i.e., the row and column k of A were interchanged with the row and column IPIV(k).

IPIV2 Output parameter.

IPIV is INTEGER array, dimension (N)

On exit, it contains the details of the interchanges, i.e., the row and column k of T were interchanged with the row and column IPIV(k).

WORK Output parameter.

WORK is REAL workspace of size LWORK

LWORK Input parameter.

The size of WORK. LWORK >= N, internally used to select NB

such that LWORK >= N*NB.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, band LU factorization failed on i-th column

Related Information

For this routine in other precisions, please see *csytrf_aa_2stage*, *dsytrf_aa_2stage* and *zsytrf_aa_2stage*. It also exists with a native C interface as *LAPACKE_ssytrf_aa_2stage*.

4.16.84 ssytri_3x

SSYTRI_3X computes the inverse of a real symmetric indefinite matrix A using the factorization computed by SSYTRF_RK or SSYTRF_BK:

```
A = P*U*D*(U**T)*(P**T) or A = P*L*D*(L**T)*(P**T),
```

where U (or L) is unit upper (or lower) triangular matrix, U^T (or L^T) is the transpose of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssytri_3x(UPLO, N, A, LDA, E, IPIV, WORK, NB, INFO)
```

C specification:

```
#include "armpl.h"

void ssytri_3x_(const char *uplo, const armpl_int_t *n, float *a,
                const armpl_int_t *lda, const float *e,
                const armpl_int_t *ipiv, float *work, const armpl_int_t *nb,
                armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, diagonal of the block diagonal matrix D and factors U or L as computed by SYTRF_RK and SSYTRF_BK: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D should be provided on entry in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

On exit, if INFO = 0, the symmetric inverse of the original matrix. If UPLO = 'U': the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; If UPLO = 'L': the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Input parameter.

E is REAL

E is an array, dimension (N). On entry, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) not referenced; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) not referenced.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is not referenced in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by SSYTRF_RK or SSYTRF_BK.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (N+NB+1,NB+3). .

NB Input parameter.

NB is INTEGER

Block size.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *csytri_3x*, *dsytri_3x* and *zsytri_3x*.

4.16.85 ssytrs_aa_2stage

SSYTRS_AA_2STAGE solves a system of linear equations A*X = B with a real symmetric matrix A using the factorization A = U*T*U^T or A = L*T*L^T computed by SSYTRF_AA_2STAGE.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssytrs_aa_2stage(UPLO, N, NRHS, A, LDA, TB, LTB, IPIV, IPIV2, B,
                            LDB, INFO)
```

C specification:

```
#include "armpl.h"

void ssytrs_aa_2stage_(const char *uplo, const armpl_int_t *n,
                       const armpl_int_t *nrhs, const float *a,
                       const armpl_int_t *lda, float *tb,
                       const armpl_int_t *ltb, const armpl_int_t *ipiv,
                       const armpl_int_t *ipiv2, float *b,
                       const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*T*U^T ; = 'L': Lower triangular, form is A = L*T*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). Details of factors computed by SSYTRF_AA_2STAGE.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TB Output parameter.

TB is REAL

TB is an array, dimension (LTB). Details of factors computed by SSYTRF_AA_2STAGE.

LTB Input parameter.

The size of the array TB. LTB >= 4*N.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges as computed by SSYTRF_AA_2STAGE.

IPIV2 Input parameter.

IPIV2 is INTEGER array, dimension (N)

Details of the interchanges as computed by SSYTRF_AA_2STAGE.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csytrs_aa_2stage*, *dsytrs_aa_2stage* and *zsytrs_aa_2stage*. It also exists with a native C interface as *LAPACKE_ssytrs_aa_2stage*.

4.16.86 strevc3

strevc3 computes some or all of the right and/or left eigenvectors of a real upper quasi-triangular matrix T. Matrices of this type are produced by the Schur factorization of a real general matrix: A = Q*T*Q^T , as computed by SHSEQR.

The right eigenvector x and the left eigenvector y of T corresponding to an eigenvalue w are defined by:

```
T*x = w*x,      (y**H)*T = w*(y**H)
```

where y^H denotes the conjugate transpose of y. The eigenvalues are not input to this routine, but are read directly from the diagonal blocks of T.

This routine returns the matrices X and/or Y of right and left eigenvectors of T, or the products Q*X and/or Q*Y, where Q is an input matrix. If Q is the orthogonal factor that reduces a matrix A to Schur form T, then Q*X and Q*Y are the matrices of right and left eigenvectors of A.

This uses a Level 3 BLAS version of the back transformation.

Syntax

Fortran specification:

```
use armpl_library

subroutine strevc3(SIDE, HOWMNY, SELECT, N, T, LDT, VL, LDVL, VR, LDVR, MM, M,
                   WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void strevc3_(const char *side, const char *howmny, armpl_int_t *select,
              const armpl_int_t *n, const float *t, const armpl_int_t *ldt,
              float *vl, const armpl_int_t *ldvl, float *vr,
              const armpl_int_t *ldvr, const armpl_int_t *mm, armpl_int_t *m,
              float *work, const armpl_int_t *lwork, armpl_int_t *info,
              ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'R': compute right eigenvectors only; = 'L': compute left eigenvectors only; = 'B': compute both right and left eigenvectors.

HOWMNY Input parameter.

HOWMNY is CHARACTER*1

= 'A': compute all right and/or left eigenvectors; = 'B': compute all right and/or left eigenvectors, backtransformed by the matrices in VR and/or VL; = 'S': compute selected right and/or left eigenvectors, as indicated by the logical array SELECT.

SELECT Input and output parameter.

SELECT is LOGICAL

SELECT is an array, dimension (N). If HOWMNY = 'S', SELECT specifies the eigenvectors to be computed. If w(j) is a real eigenvalue, the corresponding real eigenvector is computed if SELECT(j) is .TRUE.. If w(j) and w(j+1) are the real and imaginary parts of a complex eigenvalue, the corresponding complex eigenvector is computed if either SELECT(j) or SELECT(j+1) is .TRUE., and on exit SELECT(j) is set to .TRUE. and SELECT(j+1) is set to .FALSE.. Not referenced if HOWMNY = 'A' or 'B'.

N Input parameter.

N is INTEGER

The order of the matrix T. N >= 0.

T Input parameter.

T is REAL

T is an array, dimension (LDT, N). The upper quasi-triangular matrix T in Schur canonical form.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

VL Input and output parameter.

VL is REAL

VL is an array, dimension (LDVL, MM). On entry, if SIDE = 'L' or 'B' and HOWMNY = 'B', VL must contain an N-by-N matrix Q (usually the orthogonal matrix Q of Schur vectors returned by SHSEQR). On exit, if SIDE = 'L' or 'B', VL contains: if HOWMNY = 'A', the matrix Y of left eigenvectors of T; if HOWMNY = 'B', the matrix Q*Y; if HOWMNY = 'S', the left eigenvectors of T specified by SELECT, stored consecutively in the columns of VL, in the same order as their eigenvalues. A complex eigenvector corresponding to a complex eigenvalue is stored in two consecutive columns, the first holding the real part, and the second the imaginary part. Not referenced if SIDE = 'R'.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the array VL. LDVL >= 1, and if SIDE = 'L' or 'B', LDVL >= N.

VR Input and output parameter.

VR is REAL

VR is an array, dimension (LDVR, MM). On entry, if SIDE = 'R' or 'B' and HOWMNY = 'B', VR must contain an N-by-N matrix Q (usually the orthogonal matrix Q of Schur vectors returned by SHSEQR). On exit, if SIDE = 'R' or 'B', VR contains: if HOWMNY = 'A', the matrix X of right eigenvectors of T; if HOWMNY = 'B', the matrix Q*X; if HOWMNY = 'S', the right eigenvectors of T specified by SELECT, stored consecutively in the columns of VR, in the same order as their eigenvalues. A complex eigenvector corresponding to a complex eigenvalue is stored in two consecutive columns, the first holding the real part and the second the imaginary part. Not referenced if SIDE = 'L'.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the array VR. LDVR >= 1, and if SIDE = 'R' or 'B', LDVR >= N.

MM Input parameter.

MM is INTEGER

The number of columns in the arrays VL and/or VR. MM >= M.

M Output parameter.

M is INTEGER

The number of columns in the arrays VL and/or VR actually used to store the eigenvectors. If HOWMNY = 'A' or 'B', M is set to N. Each selected real eigenvector occupies one column and each selected complex eigenvector occupies two columns.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of array WORK. LWORK >= max(1,3*N). For optimum performance, LWORK >= N + 2*N*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctrevc3*, *dtrevc3* and *ztrevc3*.

4.16.87 strtrs

strtrs solves a triangular system of the form

```
A * X = B  or  A**T * X = B,
```

where A is a triangular matrix of order N, and B is an N-by-NRHS matrix. A check is made to verify that A is nonsingular.

Syntax

Fortran specification:

```
use armpl_library

subroutine strtrs(UPLO, TRANS, DIAG, N, NRHS, A, LDA, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void strtrs_(const char *uplo, const char *trans, const char *diag,
             const armpl_int_t *n, const armpl_int_t *nrhs, const float *a,
             const armpl_int_t *lda, float *b, const armpl_int_t *ldb,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose = Transpose)

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). The triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, if INFO = 0, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element of A is zero, indicating that the matrix is singular and the solutions X have not been computed.

Related Information

For this routine in other precisions, please see *ctrtrs*, *dtrtrs* and *ztrtrs*. It also exists with a native C interface as *LAPACKE_strtrs*.

4.16.88 zgetsls

zgetsls solves overdetermined or underdetermined complex linear systems involving an M-by-N matrix A, using a tall sk

The following options are provided:

* If TRANS = 'N' and m >= n: find the least squares solution of an overdetermined system, i.e., solve the least squares problem minimize || B - A*X ||.

* If TRANS = 'N' and m < n: find the minimum norm solution of an underdetermined system A * X = B.

* If TRANS = 'C' and m >= n: find the minimum norm solution of an undetermined system A^T * X = B.

* If TRANS = 'C' and m < n: find the least squares solution of an overdetermined system, i.e., solve the least squares problem minimize || B - A^T * X ||.

Several right hand side vectors b and solution vectors x can be handled in a single call; they are stored as the columns of the M-by-NRHS right hand side matrix B and the N-by-NRHS solution matrix X.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgetsls(TRANS, M, N, NRHS, A, LDA, B, LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgetsls_(const char *trans, const armpl_int_t *m, const armpl_int_t *n,
              const armpl_int_t *nrhs, armpl_doublecomplex_t *a,
              const armpl_int_t *lda, armpl_doublecomplex_t *b,
              const armpl_int_t *ldb, armpl_doublecomplex_t *work,
              const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': the linear system involves A; = 'C': the linear system involves A^H .

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >=0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, A is overwritten by details of its QR or LQ factorization as returned by ZGEQR or ZGELQ.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the matrix B of right hand side vectors, stored columnwise; B is M-by-NRHS if TRANS = 'N', or N-by-NRHS if TRANS = 'C'. On exit, if INFO = 0, B is overwritten by the solution vectors, stored columnwise: if TRANS = 'N' and m >= n, rows 1 to n of B contain the least

squares solution vectors. if TRANS = 'N' and m < n, rows 1 to N of B contain the minimum norm solution vectors; if TRANS = 'C' and m >= n, rows 1 to M of B contain the minimum norm solution vectors; if TRANS = 'C' and m < n, rows 1 to M of B contain the least squares solution vectors.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= MAX(1, M, N).

WORK Output parameter.

(workspace) COMPLEX*16

(workspace) is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) contains optimal (or either minimal or optimal, if query was assumed) LWORK. See LWORK for details.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If LWORK = -1 or -2, then a workspace query is assumed. If LWORK = -1, the routine calculates optimal size of WORK for the optimal performance and returns this value in WORK(1). If LWORK = -2, the routine calculates minimal size of WORK and returns this value in WORK(1).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element of the triangular factor of A is zero, so that A does not have full rank; the least squares solution could not be computed.

Related Information

For this routine in other precisions, please see *cgetsls*, *dgetsls* and *sgetsls*. It also exists with a native C interface as *LAPACKE_zgetsls*.

4.16.89 zhb2st_kernels

ZHB2ST_KERNELS is an internal routine used by the ZHETRD_HB2ST subroutine.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhb2st_kernels(UPLO, WANTZ, TTYPE, ST, ED, SWEEP, N, NB, IB, A,
                          LDA, V, TAU, LDVT, WORK)
```

C specification:

```
#include "armpl.h"

void zhb2st_kernels_(const char *uplo, const armpl_int_t *wantz,
                     const armpl_int_t *ttype, const armpl_int_t *st,
                     const armpl_int_t *ed, const armpl_int_t *sweep,
                     const armpl_int_t *n, const armpl_int_t *nb,
                     const armpl_int_t *ib, armpl_doublecomplex_t *a,
                     const armpl_int_t *lda, armpl_doublecomplex_t *v,
                     armpl_doublecomplex_t *tau, const armpl_int_t *ldvt,
                     const armpl_doublecomplex_t *work, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

WANTZ Input parameter.

WANTZ is LOGICAL which indicate if Eigenvalue are requested or both

Eigenvalue/Eigenvectors.

TTYPE Input parameter.

TTYPE is INTEGER

ST Input parameter.

ST is INTEGER

internal parameter for indices.

ED Input parameter.

ED is INTEGER

internal parameter for indices.

SWEEP Input parameter.

SWEEP is INTEGER

internal parameter for indices.

N Input parameter.

N is INTEGER. The order of the matrix A.

NB Input parameter.

NB is INTEGER. The size of the band.

IB Input parameter.

IB is INTEGER.

A Input and output parameter.

A is COMPLEX*16 array. A pointer to the matrix A.

LDA Input parameter.

LDA is INTEGER. The leading dimension of the matrix A.

V Output parameter.

V is COMPLEX*16

V is an array, dimension 2*n if eigenvalues only are. requested or to be queried for vectors.

TAU Output parameter.

TAU is COMPLEX*16

TAU is an array, dimension (2*n).. The scalar factors of the Householder reflectors are stored in this array.

LDVT Input parameter.

LDVT is INTEGER.

WORK Input parameter.

WORK is COMPLEX*16 array. Workspace of size nb.

Related Information

For this routine in other precisions, please see *chb2st_kernels*.

4.16.90 zhbev_2stage

ZHBEV_2STAGE computes all the eigenvalues and, optionally, eigenvectors of a complex Hermitian band matrix A using the 2stage technique for the reduction to tridiagonal.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhbev_2stage(JOBZ, UPLO, N, KD, AB, LDAB, W, Z, LDZ, WORK, LWORK,
                        RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhbev_2stage_(const char *jobz, const char *uplo, const armpl_int_t *n,
                   const armpl_int_t *kd, armpl_doublecomplex_t *ab,
                   const armpl_int_t *ldab, double *w,
                   armpl_doublecomplex_t *z, const armpl_int_t *ldz,
                   armpl_doublecomplex_t *work, const armpl_int_t *lwork,
                   double *rwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors. Not available in this release.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the Hermitian band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, AB is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the first superdiagonal and the diagonal of the tridiagonal matrix T are returned in rows KD and KD+1 of AB, and if UPLO = 'L', the diagonal and first subdiagonal of T are returned in the first two rows of AB.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD + 1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the orthonormal eigenvectors of the matrix A, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension LWORK. On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= 1, when N <= 1; otherwise If JOBZ = 'N' and N > 1, LWORK must be queried. LWORK = MAX(1, dimension) where dimension = (2KD+1)* N + KD*NTHREADS where KD is the size of the band. NTHREADS is the number of threads used when openMP compilation is enabled, otherwise =1. If JOBZ = 'V' and N > 1, LWORK must be queried. Not yet available.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (max(1,3*N-2)) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *chbev_2stage*. It also exists with a native C interface as *LAPACKE_zhbev_2stage*.

4.16.91 zhbevd_2stage

ZHBEVD_2STAGE computes all the eigenvalues and, optionally, eigenvectors of a complex Hermitian band matrix A using the 2stage technique for the reduction to tridiagonal. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhbevd_2stage(JOBZ, UPLO, N, KD, AB, LDAB, W, Z, LDZ, WORK, LWORK,
                         RWORK, LRWORK, IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhbevd_2stage_(const char *jobz, const char *uplo, const armpl_int_t *n,
                    const armpl_int_t *kd, armpl_doublecomplex_t *ab,
                    const armpl_int_t *ldab, double *w,
                    armpl_doublecomplex_t *z, const armpl_int_t *ldz,
                    armpl_doublecomplex_t *work, const armpl_int_t *lwork,
                    double *rwork, const armpl_int_t *lrwork,
                    armpl_int_t *iwork, const armpl_int_t *liwork,
                    armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors. Not available in this release.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the Hermitian band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, AB is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the first superdiagonal and the diagonal of the tridiagonal matrix T are returned in rows KD and KD+1 of AB, and if UPLO = 'L', the diagonal and first subdiagonal of T are returned in the first two rows of AB.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD + 1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, N). If JOBZ = 'V', then if INFO = 0, Z contains the orthonormal eigenvectors of the matrix A, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = 'N', then Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= 1, when N <= 1; otherwise If JOBZ = 'N' and N > 1, LWORK must be queried. LWORK = MAX(1, dimension) where dimension = (2KD+1)* N + KD*NTHREADS where KD is the size of the band. NTHREADS is the number of threads used when openMP compilation is enabled, otherwise =1. If JOBZ = 'V' and N > 1, LWORK must be queried. Not yet available.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array,. dimension (LRWORK) On exit, if INFO = 0, RWORK(1) returns the optimal LRWORK.

LRWORK Input parameter.

LRWORK is INTEGER

The dimension of array RWORK. If N <= 1, LRWORK must be at least 1. If JOBZ = 'N' and N > 1, LRWORK must be at least N. If JOBZ = 'V' and N > 1, LRWORK must be at least 1 + 5* N + 2*N**2.

If LRWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of array IWORK. If JOBZ = 'N' or N <= 1, LIWORK must be at least 1. If JOBZ = 'V' and N > 1, LIWORK must be at least 3 + 5* N .

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *chbevd_2stage*. It also exists with a native C interface as *LAPACKE_zhbevd_2stage*.

4.16.92 zhbevx_2stage

ZHBEVX_2STAGE computes selected eigenvalues and, optionally, eigenvectors of a complex Hermitian band matrix A using the 2stage technique for the reduction to tridiagonal. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhbevx_2stage(JOBZ, RANGE, UPLO, N, KD, AB, LDAB, Q, LDQ, VL, VU,
                         IL, IU, ABSTOL, M, W, Z, LDZ, WORK, LWORK, RWORK,
                         IWORK, IFAIL, INFO)
```

C specification:

```
#include "armpl.h"

void zhbevx_2stage_(const char *jobz, const char *range, const char *uplo,
                    const armpl_int_t *n, const armpl_int_t *kd,
                    armpl_doublecomplex_t *ab, const armpl_int_t *ldab,
                    armpl_doublecomplex_t *q, const armpl_int_t *ldq,
                    const double *vl, const double *vu, const armpl_int_t *il,
                    const armpl_int_t *iu, const double *abstol,
                    armpl_int_t *m, double *w, armpl_doublecomplex_t *z,
                    const armpl_int_t *ldz, armpl_doublecomplex_t *work,
                    const armpl_int_t *lwork, double *rwork,
                    armpl_int_t *iwork, armpl_int_t *ifail, armpl_int_t *info,
                    ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors. Not available in this release.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found; = 'V': all eigenvalues in the half-open interval (VL,VU] will be found; = 'I': the IL-th through IU-th eigenvalues will be found.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the Hermitian band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, AB is overwritten by values generated during the reduction to tridiagonal form.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD + 1.

Q Output parameter.

Q is COMPLEX*16

Q is an array, dimension (LDQ, N). If JOBZ = 'V', the N-by-N unitary matrix used in the reduction to tridiagonal form. If JOBZ = 'N', the array Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. If JOBZ = 'V', then LDQ >= max(1, N).

VL Input parameter.

VL is DOUBLE PRECISION

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is DOUBLE PRECISION

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is DOUBLE PRECISION

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing AB to tridiagonal form.

Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*DLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*DLAMCH('S').

See "Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy," by Demmel and Kahan, LAPACK Working Note #3.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). The first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). If an eigenvector fails to converge, then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL. If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= 1, when N <= 1; otherwise If JOBZ = 'N' and N > 1, LWORK must be queried. LWORK = MAX(1, dimension) where dimension = (2KD+1)* N + KD*NTHREADS where KD is the size of the band. NTHREADS is the number of threads used when openMP compilation is enabled, otherwise =1. If JOBZ = 'V' and N > 1, LWORK must be queried. Not yet available.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (7*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (5*N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (N)

If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, then i eigenvectors failed to converge. Their indices are stored in array IFAIL.

Related Information

For this routine in other precisions, please see *chbevx_2stage*. It also exists with a native C interface as *LAPACKE_zhbevx_2stage*.

4.16.93 zheev_2stage

ZHEEV_2STAGE computes all eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix A using the 2stage technique for the reduction to tridiagonal.

Syntax

Fortran specification:

```
use armpl_library

subroutine zheev_2stage(JOBZ, UPLO, N, A, LDA, W, WORK, LWORK, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zheev_2stage_(const char *jobz, const char *uplo, const armpl_int_t *n,
                   armpl_doublecomplex_t *a, const armpl_int_t *lda,
                   double *w, armpl_doublecomplex_t *work,
                   const armpl_int_t *lwork, double *rwork, armpl_int_t *info,
                   ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors. Not available in this release.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A. On exit, if JOBZ = 'V', then if INFO = 0, A contains the orthonormal eigenvectors of the matrix A. If JOBZ = 'N', then on exit the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= 1, when N <= 1; otherwise If JOBZ = 'N' and N > 1, LWORK must be queried. LWORK = MAX(1, dimension) where dimension = max(stage1,stage2) + (KD+1)* N + N = N*KD + N*max(KD+1,FACTOPTNB) + max(2*KD*KD, KD*NTHREADS) + (KD+1)* N + N where KD is the blocking size of the reduction, FACTOPTNB is the blocking used by the QR or LQ algorithm, usually FACTOPTNB=128 is a good choice NTHREADS is the number of threads used when openMP compilation is enabled, otherwise =1. If JOBZ = 'V' and N > 1, LWORK must be queried. Not yet available

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (max(1, 3*N-2)) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

Related Information

For this routine in other precisions, please see *cheev_2stage*. It also exists with a native C interface as *LAPACKE_zheev_2stage*.

4.16.94 zheevd_2stage

ZHEEVD_2STAGE computes all eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix A using the 2stage technique for the reduction to tridiagonal. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine zheevd_2stage(JOBZ, UPLO, N, A, LDA, W, WORK, LWORK, RWORK, LRWORK,
                         IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zheevd_2stage_(const char *jobz, const char *uplo, const armpl_int_t *n,
                    armpl_doublecomplex_t *a, const armpl_int_t *lda,
                    double *w, armpl_doublecomplex_t *work,
                    const armpl_int_t *lwork, double *rwork,
                    const armpl_int_t *lrwork, armpl_int_t *iwork,
                    const armpl_int_t *liwork, armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors. Not available in this release.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A. On exit, if JOBZ = 'V', then if INFO = 0, A contains the orthonormal eigenvectors of the matrix A. If JOBZ = 'N', then on exit the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If N <= 1, LWORK must be at least 1. If JOBZ = 'N' and N > 1, LWORK must be queried. LWORK = MAX(1, dimension) where dimension = max(stage1,stage2) + (KD+1)* N + N+1 = N*KD + N*max(KD+1,FACTOPTNB) + max(2*KD*KD, KD*NTHREADS) + (KD+1)* N + N+1 where KD is the blocking size of the reduction, FACTOPTNB is the blocking used by the QR or LQ algorithm, usually FACTOPTNB=128 is a good choice NTHREADS is the number of threads used when openMP compilation is enabled, otherwise =1. If JOBZ = 'V' and N > 1, LWORK must be at least 2* N + N**2

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array,. dimension (LRWORK) On exit, if INFO = 0, RWORK(1) returns the optimal LRWORK.

LRWORK Input parameter.

LRWORK is INTEGER

The dimension of the array RWORK. If N <= 1, LRWORK must be at least 1. If JOBZ = 'N' and N > 1, LRWORK must be at least N. If JOBZ = 'V' and N > 1, LRWORK must be at least 1 + 5* N + 2*N**2.

If LRWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. If N <= 1, LIWORK must be at least 1. If JOBZ = 'N' and N > 1, LIWORK must be at least 1. If JOBZ = 'V' and N > 1, LIWORK must be at least 3 + 5*N.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i and JOBZ = 'N', then the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero; if INFO = i and JOBZ = 'V', then the algorithm failed to compute an eigenvalue while working on the submatrix lying in rows and columns INFO/(N+1) through mod(INFO,N+1).

Related Information

For this routine in other precisions, please see *cheevd_2stage*. It also exists with a native C interface as *LAPACKE_zheevd_2stage*.

4.16.95 zheevr_2stage

ZHEEVR_2STAGE computes selected eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix A using the 2stage technique for the reduction to tridiagonal. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

ZHEEVR_2STAGE first reduces the matrix A to tridiagonal form T with a call to ZHETRD. Then, whenever possible, ZHEEVR_2STAGE calls ZSTEMR to compute eigenspectrum using Relatively Robust Representations. ZSTEMR computes eigenvalues by the dqds algorithm, while orthogonal eigenvectors are computed from various "good" L D L^T representations (also known as Relatively Robust Representations). Gram-Schmidt orthogonalization is avoided as far as possible. More specifically, the various steps of the algorithm are as follows.

For each unreduced block (submatrix) of T,

```
(a) Compute T - sigma I  = L D L^T, so that L and D
    define all the wanted eigenvalues to high relative accuracy.
    This means that small relative changes in the entries of D and L
    cause only small relative changes in the eigenvalues and
    eigenvectors. The standard (unfactored) representation of the
    tridiagonal matrix T does not have this property in general.
(b) Compute the eigenvalues to suitable accuracy.
    If the eigenvectors are desired, the algorithm attains full
    accuracy of the computed eigenvalues only right before
    the corresponding vectors have to be computed, see steps c) and d).
(c) For each cluster of close eigenvalues, select a new
    shift close to the cluster, find a new factorization, and refine
    the shifted eigenvalues to suitable accuracy.
(d) For each eigenvalue with a large enough relative separation compute
    the corresponding eigenvector by forming a rank revealing twisted
    factorization. Go back to (c) for any clusters that remain.
```

The desired accuracy of the output can be specified by the input parameter ABSTOL.

For more details, see DSTEMR's documentation and: - Inderjit S. Dhillon and Beresford N. Parlett: "Multiple representations

```
 to compute orthogonal eigenvectors of symmetric tridiagonal matrices,"
Linear Algebra and its Applications, 387(1), pp. 1-28, August 2004.
```

- Inderjit Dhillon and Beresford Parlett: "Orthogonal Eigenvectors and

```
 Relative Gaps," SIAM Journal on Matrix Analysis and Applications, Vol. 25,
2004.  Also LAPACK Working Note 154.
```

- Inderjit Dhillon: "A new O(n^2) algorithm for the symmetric

```
 tridiagonal eigenvalue/eigenvector problem",
Computer Science Division Technical Report No. UCB/CSD-97-971,
UC Berkeley, May 1997.
```

Note 1 : ZHEEVR_2STAGE calls ZSTEMR when the full spectrum is requested on machines which conform to the ieee-754 floating point standard. ZHEEVR_2STAGE calls DSTEBZ and ZSTEIN on non-ieee machines and when partial spectrum requests are made.

Normal execution of ZSTEMR may create NaNs and infinities and hence may abort due to a floating point exception in environments which do not handle NaNs and infinities in the ieee standard default manner.

Syntax

Fortran specification:

```
use armpl_library

subroutine zheevr_2stage(JOBZ, RANGE, UPLO, N, A, LDA, VL, VU, IL, IU, ABSTOL,
                         M, W, Z, LDZ, ISUPPZ, WORK, LWORK, RWORK, LRWORK,
                         IWORK, LIWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zheevr_2stage_(const char *jobz, const char *range, const char *uplo,
                    const armpl_int_t *n, armpl_doublecomplex_t *a,
                    const armpl_int_t *lda, const double *vl,
                    const double *vu, const armpl_int_t *il,
                    const armpl_int_t *iu, const double *abstol,
                    armpl_int_t *m, double *w, armpl_doublecomplex_t *z,
                    const armpl_int_t *ldz, armpl_int_t *isuppz,
                    armpl_doublecomplex_t *work, const armpl_int_t *lwork,
                    double *rwork, const armpl_int_t *lrwork,
                    armpl_int_t *iwork, const armpl_int_t *liwork,
                    armpl_int_t *info, ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors. Not available in this release.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found. = 'V': all eigenvalues in the half-open interval (VL,VU] will be found. = 'I': the IL-th through IU-th eigenvalues will be found. For RANGE = 'V' or 'I' and IU - IL < N - 1, DSTEBZ and ZSTEIN are called

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A. On exit, the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

VL Input parameter.

VL is DOUBLE PRECISION

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is DOUBLE PRECISION

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is DOUBLE PRECISION

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing A to tridiagonal form.

See “Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy,” by Demmel and Kahan, LAPACK Working Note #3.

If high relative accuracy is important, set ABSTOL to DLAMCH(‘Safe minimum’). Doing so will guarantee that eigenvalues are computed to high relative accuracy when possible in future releases. The current code does not make any guarantees about high relative accuracy, but furutre releases will. See J. Barlow and J. Demmel, “Computing Accurate Eigensystems of Scaled Diagonally Dominant Matrices”, LAPACK Working Note #7, for a discussion of which matrices define their eigenvalues to high relative accuracy.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = ‘A’, M = N, and if RANGE = ‘I’, M = IU-IL+1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). The first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = ‘V’, then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). If JOBZ = ‘N’, then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = ‘V’, the exact value of M is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = ‘V’, LDZ >= max(1, N).

ISUPPZ Output parameter.

ISUPPZ is INTEGER array, dimension (2*max(1, M))

The support of the eigenvectors in Z, i.e., the indices indicating the nonzero elements in Z. The i-th eigenvector is nonzero only in elements ISUPPZ(2*i-1) through ISUPPZ(2*i). This is an output of ZSTEMR (tridiagonal matrix). The support of the eigenvectors of A is typically 1:N because of the unitary transformations applied by ZUNMTR. Implemented only for RANGE = ‘A’ or ‘I’ and IU - IL = N - 1

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If JOBZ = ‘N’ and N > 1, LWORK must be queried. LWORK = MAX(1, 26*N, dimension) where dimension = max(stage1,stage2) + (KD+1)* N + N = N*KD + N*max(KD+1,FACTOPTNB) + max(2*KD*KD, KD*NTHREADS) + (KD+1)* N + N where KD is the blocking size of the reduction, FACTOPTNB is the blocking used by the QR or LQ algorithm, usually FACTOPTNB=128 is a good choice NTHREADS is the number of threads used when openMP compilation is enabled, otherwise =1. If JOBZ = ‘V’ and N > 1, LWORK must be queried. Not yet available

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (MAX(1, LRWORK)). On exit, if INFO = 0, RWORK(1) returns the optimal (and minimal) LRWORK.

LRWORK Input parameter.

LRWORK is INTEGER

The length of the array RWORK. LRWORK >= max(1,24*N).

If LRWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MAX(1, LIWORK))

On exit, if INFO = 0, IWORK(1) returns the optimal (and minimal) LIWORK.

LIWORK Input parameter.

LIWORK is INTEGER

The dimension of the array IWORK. LIWORK >= max(1,10*N).

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal sizes of the WORK, RWORK and IWORK arrays, returns these values as the first entries of the WORK, RWORK and IWORK arrays, and no error message related to LWORK or LRWORK or LIWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: Internal error

Related Information

For this routine in other precisions, please see *cheevr_2stage*. It also exists with a native C interface as *LAPACKE_zheevr_2stage*.

4.16.96 zheevx_2stage

ZHEEVX_2STAGE computes selected eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix A using the 2stage technique for the reduction to tridiagonal. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine zheevx_2stage(JOBZ, RANGE, UPLO, N, A, LDA, VL, VU, IL, IU, ABSTOL,
                         M, W, Z, LDZ, WORK, LWORK, RWORK, IWORK, IFAIL,
                         INFO)
```

C specification:

```
#include "armpl.h"

void zheevx_2stage_(const char *jobz, const char *range, const char *uplo,
                    const armpl_int_t *n, armpl_doublecomplex_t *a,
                    const armpl_int_t *lda, const double *vl,
                    const double *vu, const armpl_int_t *il,
                    const armpl_int_t *iu, const double *abstol,
                    armpl_int_t *m, double *w, armpl_doublecomplex_t *z,
                    const armpl_int_t *ldz, armpl_doublecomplex_t *work,
                    const armpl_int_t *lwork, double *rwork,
                    armpl_int_t *iwork, armpl_int_t *ifail, armpl_int_t *info,
                    ... );
```

Parameters

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors. Not available in this release.

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': all eigenvalues will be found. = 'V': all eigenvalues in the half-open interval (VL,VU] will be found. = 'I': the IL-th through IU-th eigenvalues will be found.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A. On exit, the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

VL Input parameter.

VL is DOUBLE PRECISION

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is DOUBLE PRECISION

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL Input parameter.

ABSTOL is DOUBLE PRECISION

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing A to tridiagonal form.

Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*DLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*DLAMCH('S').

See "Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy," by Demmel and Kahan, LAPACK Working Note #3.

M Output parameter.

M is INTEGER

The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). On normal exit, the first M elements contain the selected eigenvalues in ascending order.

Z Output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, max(1, M)). If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). If an eigenvector fails to converge, then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL. If JOBZ = 'N', then Z is not referenced. Note: the user must ensure that at least max(1, M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= 1, when N <= 1; otherwise If JOBZ = 'N' and N > 1, LWORK must be queried. LWORK = MAX(1, 8*N, dimension) where dimension = max(stage1,stage2) + (KD+1)* N + N = N*KD + N*max(KD+1,FACTOPTNB) + max(2*KD*KD, KD*NTHREADS) + (KD+1)* N + N where KD is the blocking size of the reduction, FACTOPTNB is the blocking used by the QR or LQ algorithm, usually FACTOPTNB=128 is a good choice NTHREADS is the number of threads used when openMP compilation is enabled, otherwise =1. If JOBZ = 'V' and N > 1, LWORK must be queried. Not yet available

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (7*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (5*N)

IFAIL Output parameter.

IFAIL is INTEGER array, dimension (N)

If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, then i eigenvectors failed to converge. Their indices are stored in array IFAIL.

Related Information

For this routine in other precisions, please see *cheevx_2stage*. It also exists with a native C interface as *LAPACKE_zheevx_2stage*.

4.16.97 zhegv_2stage

ZHEGV_2STAGE computes all the eigenvalues, and optionally, the eigenvectors of a complex generalized Hermitian-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x. Here A and B are assumed to be Hermitian and B is also positive definite. This routine use the 2stage technique for the reduction to tridiagonal which showed higher performance on recent architecture and for large sizes N>2000.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhegv_2stage(ITYPE, JOBZ, UPLO, N, A, LDA, B, LDB, W, WORK, LWORK,
                        RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhegv_2stage_(const armpl_int_t *itype, const char *jobz,
                   const char *uplo, const armpl_int_t *n,
                   armpl_doublecomplex_t *a, const armpl_int_t *lda,
                   armpl_doublecomplex_t *b, const armpl_int_t *ldb,
                   double *w, armpl_doublecomplex_t *work,
                   const armpl_int_t *lwork, double *rwork, armpl_int_t *info,
                   ... );
```

Parameters

ITYPE Input parameter.

ITYPE is INTEGER

Specifies the problem type to be solved: = 1: A*x = (lambda)*B*x = 2: A*B*x = (lambda)*x = 3: B*A*x = (lambda)*x

JOBZ Input parameter.

JOBZ is CHARACTER*1

= 'N': Compute eigenvalues only; = 'V': Compute eigenvalues and eigenvectors. Not available in this release.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangles of A and B are stored; = 'L': Lower triangles of A and B are stored.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A.

On exit, if JOBZ = 'V', then if INFO = 0, A contains the matrix Z of eigenvectors. The eigenvectors are normalized as follows: if ITYPE = 1 or 2, Z^H *B*Z = I; if ITYPE = 3, Z^H *inv(B)*Z = I. If JOBZ = 'N', then on exit the upper triangle (if UPLO='U') or the lower triangle (if UPLO='L') of A, including the diagonal, is destroyed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). On entry, the Hermitian positive definite matrix B. If UPLO = 'U', the leading N-by-N upper triangular part of B contains the upper triangular part of the matrix B. If UPLO = 'L', the leading N-by-N lower triangular part of B contains the lower triangular part of the matrix B.

On exit, if INFO <= N, the part of B containing the matrix is overwritten by the triangular factor U or L from the Cholesky factorization B = U^H *U or B = L*L^H .

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). If INFO = 0, the eigenvalues in ascending order.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)). On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK Input parameter.

LWORK is INTEGER

The length of the array WORK. LWORK >= 1, when N <= 1; otherwise If JOBZ = 'N' and N > 1, LWORK must be queried. LWORK = MAX(1, dimension) where dimension = max(stage1,stage2) + (KD+1)* N + N = N*KD + N*max(KD+1,FACTOPTNB) + max(2*KD*KD, KD*NTHREADS) + (KD+1)* N + N where KD is the blocking size of the reduction, FACTOPTNB is the blocking used by the QR or LQ algorithm, usually FACTOPTNB=128 is a good choice NTHREADS is the number of threads used when openMP compilation is enabled, otherwise =1. If JOBZ = 'V' and N > 1, LWORK must be queried. Not yet available

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (max(1, 3*N-2)) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: ZPOTRF or ZHEEV returned an error code: <= N: if INFO = i, ZHEEV failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero; > N: if INFO = N + i, for 1 <= i <= N, then the leading minor of order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

Related Information

For this routine in other precisions, please see *chegv_2stage*. It also exists with a native C interface as *LAPACKE_zhegv_2stage*.

4.16.98 zhesv_aa_2stage

ZHESV_AA_2STAGE computes the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N Hermitian matrix and X and B are N-by-NRHS matrices.

Aasen's 2-stage algorithm is used to factor A as

```
A = U * T * U**H,  if UPLO = 'U', or
A = L * T * L**H,  if UPLO = 'L',
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and T is Hermitian and band. The matrix T is then LU-factored with partial pivoting. The factored form of A is then used to solve the system of equations A * X = B.

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhesv_aa_2stage(UPLO, N, NRHS, A, LDA, TB, LTB, IPIV, IPIV2, B,
                           LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhesv_aa_2stage_(const char *uplo, const armpl_int_t *n,
                      const armpl_int_t *nrhs, armpl_doublecomplex_t *a,
                      const armpl_int_t *lda, armpl_doublecomplex_t *tb,
                      const armpl_int_t *ltb, armpl_int_t *ipiv,
                      armpl_int_t *ipiv2, armpl_doublecomplex_t *b,
                      const armpl_int_t *ldb, armpl_doublecomplex_t *work,
                      const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, L is stored below (or above) the subdiaonal blocks, when UPLO is 'L' (or 'U').

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TB Output parameter.

TB is COMPLEX*16

TB is an array, dimension (LTB). On exit, details of the LU factorization of the band matrix.

LTB Input parameter.

The size of the array TB. LTB >= 4*N, internally

used to select NB such that LTB >= (3*NB+1)*N.

If LTB = -1, then a workspace query is assumed; the routine only calculates the optimal size of LTB, returns this value as the first entry of TB, and no error message related to LTB is issued by XERBLA.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

On exit, it contains the details of the interchanges, i.e., the row and column k of A were interchanged with the row and column IPIV(k).

IPIV2 Output parameter.

IPIV is INTEGER array, dimension (N)

On exit, it contains the details of the interchanges, i.e., the row and column k of T were interchanged with the row and column IPIV(k).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Output parameter.

WORK is COMPLEX*16 workspace of size LWORK

LWORK Input parameter.

The size of WORK. LWORK >= N, internally used to select NB

such that LWORK >= N*NB.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, band LU factorization failed on i-th column

Related Information

For this routine in other precisions, please see *chesv_aa_2stage*. It also exists with a native C interface as *LAPACKE_zhesv_aa_2stage*.

4.16.99 zhetf2_rk

ZHETF2_RK computes the factorization of a complex Hermitian matrix A using the bounded Bunch-Kaufman (rook) diagonal pivoting method:

```
A = P*U*D*(U**H)*(P**T) or A = P*L*D*(L**H)*(P**T),
```

where U (or L) is unit upper (or lower) triangular matrix, U^H (or L^H) is the conjugate of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is Hermitian and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the unblocked version of the algorithm, calling Level 2 BLAS. For more information see Further Details section.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhetf2_rk(UPLO, N, A, LDA, E, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void zhetf2_rk_(const char *uplo, const armpl_int_t *n,
                armpl_doublecomplex_t *a, const armpl_int_t *lda,
                armpl_doublecomplex_t *e, armpl_int_t *ipiv,
                armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the Hermitian matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U': the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced.

If UPLO = 'L': the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, contains: a) ONLY diagonal elements of the Hermitian block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Output parameter.

E is COMPLEX*16

E is an array, dimension (N). On exit, contains the superdiagonal (or subdiagonal) elements of the Hermitian block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) is set to 0; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) is set to 0.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is set to 0 in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

IPIV describes the permutation matrix P in the factorization of matrix A as follows. The absolute value of IPIV(k) represents the index of row and column that were interchanged with the k-th row and column. The value of UPLO describes the order in which the interchanges were applied. Also, the sign of IPIV represents the block structure of the Hermitian block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks which correspond to 1 or 2 interchanges at each factorization step. For more info see Further Details section.

If UPLO = 'U', (in factorization order, k decreases from N to 1): a) A single positive entry IPIV(k) > 0 means: D(k,k) is a 1-by-1 diagonal block. If IPIV(k) != k, rows and columns k and IPIV(k) were interchanged in the matrix A(1:N,1:N); If IPIV(k) = k, no interchange occurred.

b) A pair of consecutive negative entries IPIV(k) < 0 and IPIV(k-1) < 0 means: D(k-1:k,k-1:k) is a 2-by-2 diagonal block. (NOTE: negative entries in IPIV appear ONLY in pairs). 1) If -IPIV(k) != k, rows and columns k and -IPIV(k) were interchanged in the matrix A(1:N,1:N). If -IPIV(k) = k, no interchange occurred. 2) If -IPIV(k-1) != k-1, rows and columns k-1 and -IPIV(k-1) were interchanged in the matrix A(1:N,1:N). If -IPIV(k-1) = k-1, no interchange occurred.

c) In both cases a) and b), always ABS(IPIV(k)) <= k.

d) NOTE: Any entry IPIV(k) is always NONZERO on output.

If UPLO = 'L', (in factorization order, k increases from 1 to N): a) A single positive entry IPIV(k) > 0 means: D(k,k) is a 1-by-1 diagonal block. If IPIV(k) != k, rows and columns k and IPIV(k) were interchanged in the matrix A(1:N,1:N). If IPIV(k) = k, no interchange occurred.

b) A pair of consecutive negative entries IPIV(k) < 0 and IPIV(k+1) < 0 means: D(k:k+1,k:k+1) is a 2-by-2 diagonal block. (NOTE: negative entries in IPIV appear ONLY in pairs). 1) If -IPIV(k) != k, rows and columns k and -IPIV(k) were interchanged in the matrix A(1:N,1:N). If -IPIV(k) = k, no interchange occurred. 2) If -IPIV(k+1) != k+1, rows and columns k-1 and -IPIV(k-1) were interchanged in the matrix A(1:N,1:N). If -IPIV(k+1) = k+1, no interchange occurred.

c) In both cases a) and b), always ABS(IPIV(k)) >= k.

d) NOTE: Any entry IPIV(k) is always NONZERO on output.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit

< 0: If INFO = -k, the k-th argument had an illegal value

> 0: If INFO = k, the matrix A is singular, because: If UPLO = 'U': column k in the upper triangular part of A contains all zeros. If UPLO = 'L': column k in the lower triangular part of A contains all zeros.

Therefore D(k,k) is exactly zero, and superdiagonal elements of column k of U (or subdiagonal elements of column k of L) are all zeros. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

NOTE: INFO only stores the first occurrence of a singularity, any subsequent occurrence of singularity is not stored in INFO even though the factorization always completes.

Related Information

For this routine in other precisions, please see *chetf2_rk*.

4.16.100 zhetrd_2stage

ZHETRD_2STAGE reduces a complex Hermitian matrix A to real symmetric tridiagonal form T by a unitary similarity transformation: $Q1^H$ $Q2^H$ * A * Q2 * Q1 = T.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhetrd_2stage(VECT, UPLO, N, A, LDA, D, E, TAU, HOUS2, LHOUS2,
                         WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhetrd_2stage_(const char *vect, const char *uplo, const armpl_int_t *n,
                    armpl_doublecomplex_t *a, const armpl_int_t *lda,
                    double *d, double *e, armpl_doublecomplex_t *tau,
                    armpl_doublecomplex_t *hous2, const armpl_int_t *lhous2,
                    armpl_doublecomplex_t *work, const armpl_int_t *lwork,
                    armpl_int_t *info, ... );
```

Parameters

VECT Input parameter.

VECT is CHARACTER*1

= 'N': No need for the Housholder representation, in particular for the second stage (Band to tridiagonal) and thus LHOUS2 is of size max(1, 4*N); = 'V': the Householder representation is needed to either generate Q1 Q2 or to apply Q1 Q2, then LHOUS2 is to be queried and computed. (NOT AVAILABLE IN THIS RELEASE).

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, if UPLO = 'U', the band superdiagonal of A are overwritten by the corresponding elements of the internal band-diagonal matrix AB, and the elements above the KD superdiagonal, with the array TAU, represent the unitary matrix

Q1 as a product of elementary reflectors; if UPLO = 'L', the diagonal and band subdiagonal of A are overwritten by the corresponding elements of the internal band-diagonal matrix AB, and the elements below the KD subdiagonal, with the array TAU, represent the unitary matrix Q1 as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

D Output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The diagonal elements of the tridiagonal matrix T.

E Output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N-1). The off-diagonal elements of the tridiagonal matrix T.

TAU Output parameter.

TAU is COMPLEX*16

TAU is an array, dimension (N-KD). The scalar factors of the elementary reflectors of the first stage (see Further Details).

HOUS2 Output parameter.

HOUS2 is COMPLEX*16

HOUS2 is an array, dimension LHOUS2, that. store the Householder representation of the stage2 band to tridiagonal.

LHOUS2 Input parameter.

LHOUS2 is INTEGER

The dimension of the array HOUS2. LHOUS2 = MAX(1, dimension) If LWORK = -1, or LHOUS2=-1, then a query is assumed; the routine only calculates the optimal size of the HOUS2 array, returns this value as the first entry of the HOUS2 array, and no error message related to LHOUS2 is issued by XERBLA. LHOUS2 = MAX(1, dimension) where dimension = 4* N if VECT='N' not available now if VECT='H'

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK = MAX(1, dimension) If LWORK = -1, or LHOUS2=-1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA. LWORK = MAX(1, dimension) where dimension = max(stage1,stage2) + (KD+1)*N = N*KD + N*max(KD+1,FACTOPTNB) + max(2*KD*KD, KD*NTHREADS) + (KD+1)* N where KD is the blocking size of the reduction, FACTOPTNB is the blocking used by the QR or LQ algorithm, usually FACTOPTNB=128 is a good choice NTHREADS is the number of threads used when openMP compilation is enabled, otherwise =1.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *chetrd_2stage*.

4.16.101 zhetrd_he2hb

ZHETRD_HE2HB reduces a complex Hermitian matrix A to complex Hermitian band-diagonal form AB by a unitary similarity transformation: Q^H * A * Q = AB.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhetrd_he2hb(UPLO, N, KD, A, LDA, AB, LDAB, TAU, WORK, LWORK,
                        INFO)
```

C specification:

```
#include "armpl.h"

void zhetrd_he2hb_(const char *uplo, const armpl_int_t *n,
                   const armpl_int_t *kd, armpl_doublecomplex_t *a,
                   const armpl_int_t *lda, armpl_doublecomplex_t *ab,
                   const armpl_int_t *ldab, armpl_doublecomplex_t *tau,
                   armpl_doublecomplex_t *work, const armpl_int_t *lwork,
                   armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of superdiagonals of the reduced matrix if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0. The reduced matrix is stored in the array AB.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, if UPLO = 'U', the diagonal and first superdiagonal of A are overwritten by the corresponding elements of the tridiagonal matrix T, and the elements above the first superdiagonal, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors; if UPLO = 'L', the diagonal and first subdiagonal of A are over- written

by the corresponding elements of the tridiagonal matrix T, and the elements below the first subdiagonal, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AB Output parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). On exit, the upper or lower triangle of the Hermitian band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

TAU Output parameter.

TAU is COMPLEX*16

TAU is an array, dimension (N-KD). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (LWORK). On exit, if INFO = 0, or if LWORK=-1, WORK(1) returns the size of LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK which should be calculated by a workspace query. LWORK = MAX(1, LWORK_QUERY) If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA. LWORK_QUERY = N* KD + N*max(KD,FACTOPTNB) + 2*KD*KD where FACTOPTNB is the blocking used by the QR or LQ algorithm, usually FACTOPTNB=128 is a good choice otherwise putting LWORK=-1 will provide the size of WORK.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *chetrd_he2hb*.

4.16.102 zhetrf_aa_2stage

ZHETRF_AA_2STAGE computes the factorization of a double hermitian matrix A using the Aasen's algorithm. The form of the factorization is

```
A = U*T*U**T  or  A = L*T*L**T
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and T is a hermitian band matrix with the bandwidth of NB (NB is internally selected and stored in TB(1), and T is LU factorized with partial pivoting).

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhetrf_aa_2stage(UPLO, N, A, LDA, TB, LTB, IPIV, IPIV2, WORK,
                            LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zhetrf_aa_2stage_(const char *uplo, const armpl_int_t *n,
                       armpl_doublecomplex_t *a, const armpl_int_t *lda,
                       armpl_doublecomplex_t *tb, const armpl_int_t *ltb,
                       armpl_int_t *ipiv, armpl_int_t *ipiv2,
                       armpl_doublecomplex_t *work, const armpl_int_t *lwork,
                       armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, L is stored below (or above) the subdiaonal blocks, when UPLO is 'L' (or 'U').

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TB Output parameter.

TB is COMPLEX

TB is an array, dimension (LTB). On exit, details of the LU factorization of the band matrix.

LTB Input parameter.

The size of the array TB. LTB >= 4*N, internally

used to select NB such that LTB >= (3*NB+1)*N.

If LTB = -1, then a workspace query is assumed; the routine only calculates the optimal size of LTB, returns this value as the first entry of TB, and no error message related to LTB is issued by XERBLA.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

On exit, it contains the details of the interchanges, i.e., the row and column k of A were interchanged with the row and column IPIV(k).

IPIV2 Output parameter.

IPIV is INTEGER array, dimension (N)

On exit, it contains the details of the interchanges, i.e., the row and column k of T were interchanged with the row and column IPIV(k).

WORK Output parameter.

WORK is COMPLEX workspace of size LWORK

LWORK Input parameter.

The size of WORK. LWORK >= N, internally used to select NB

such that LWORK >= N*NB.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, band LU factorization failed on i-th column

Related Information

For this routine in other precisions, please see *chetrf_aa_2stage*. It also exists with a native C interface as *LAPACKE_zhetrf_aa_2stage*.

4.16.103 zhetri_3x

ZHETRI_3X computes the inverse of a complex Hermitian indefinite matrix A using the factorization computed by ZHETRF_RK or ZHETRF_BK:

```
A = P*U*D*(U**H)*(P**T) or A = P*L*D*(L**H)*(P**T),
```

where U (or L) is unit upper (or lower) triangular matrix, U^H (or L^H) is the conjugate of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is Hermitian and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhetri_3x(UPLO, N, A, LDA, E, IPIV, WORK, NB, INFO)
```

C specification:

```
#include "armpl.h"

void zhetri_3x_(const char *uplo, const armpl_int_t *n,
                armpl_doublecomplex_t *a, const armpl_int_t *lda,
                const armpl_doublecomplex_t *e, const armpl_int_t *ipiv,
                armpl_doublecomplex_t *work, const armpl_int_t *nb,
                armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, diagonal of the block diagonal matrix D and factors U or L as computed by ZHETRF_RK and ZHETRF_BK: a) ONLY diagonal elements of the Hermitian block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D should be provided on entry in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

On exit, if INFO = 0, the Hermitian inverse of the original matrix. If UPLO = 'U': the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; If UPLO = 'L': the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Input parameter.

E is COMPLEX*16

E is an array, dimension (N). On entry, contains the superdiagonal (or subdiagonal) elements of the Hermitian block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) not referenced; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) not referenced.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is not referenced in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZHETRF_RK or ZHETRF_BK.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N+NB+1,NB+3). .

NB Input parameter.

NB is INTEGER

Block size.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *chetri_3x*.

4.16.104 zhetrs_aa_2stage

ZHETRS_AA_2STAGE solves a system of linear equations A*X = B with a hermitian matrix A using the factorization A = U*T*U^T or A = L*T*L^T computed by ZHETRF_AA_2STAGE.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhetrs_aa_2stage(UPLO, N, NRHS, A, LDA, TB, LTB, IPIV, IPIV2, B,
                            LDB, INFO)
```

C specification:

```
#include "armpl.h"

void zhetrs_aa_2stage_(const char *uplo, const armpl_int_t *n,
                       const armpl_int_t *nrhs,
                       const armpl_doublecomplex_t *a, const armpl_int_t *lda,
                       armpl_doublecomplex_t *tb, const armpl_int_t *ltb,
                       const armpl_int_t *ipiv, const armpl_int_t *ipiv2,
                       armpl_doublecomplex_t *b, const armpl_int_t *ldb,
                       armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*T*U^T ; = 'L': Lower triangular, form is A = L*T*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is COMPLEX*16``array, dimension (``LDA, N)

Details of factors computed by ZHETRF_AA_2STAGE.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TB Output parameter.

TB is COMPLEX*16``array, dimension (``LTB)

Details of factors computed by ZHETRF_AA_2STAGE.

LTB Input parameter.

The size of the array TB. LTB >= 4*N.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges as computed by ZHETRF_AA_2STAGE.

IPIV2 Input parameter.

IPIV2 is INTEGER array, dimension (N)

Details of the interchanges as computed by ZHETRF_AA_2STAGE.

B Input and output parameter.

B is COMPLEX*16``array, dimension (``LDB, NRHS)

On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *chetrs_aa_2stage*. It also exists with a native C interface as *LAPACKE_zhetrs_aa_2stage*.

4.16.105 zhptrs

zhptrs solves a system of linear equations A*X = B with a complex Hermitian matrix A stored in packed format using the factorization A = U*D*U^H or A = L*D*L^H computed by ZHPTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhptrs(UPLO, N, NRHS, AP, IPIV, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void zhptrs_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_doublecomplex_t *ap, const armpl_int_t *ipiv,
             armpl_doublecomplex_t *b, const armpl_int_t *ldb,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^H ; = 'L': Lower triangular, form is A = L*D*L^H .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

AP Input parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by ZHPTRF, stored as a packed triangular matrix.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZHPTRF.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *chptrs*. It also exists with a native C interface as *LAPACKE_zhptrs*.

4.16.106 zlahef_rk

ZLAHEF_RK computes a partial factorization of a complex Hermitian matrix A using the bounded Bunch-Kaufman (rook) diagonal pivoting method. The partial factorization has the form:

A = (I U12) (A11 0) (I 0) if UPLO = 'U', or:

```
( 0  U22 ) (  0   D  ) ( U12**H U22**H )
```

A = (L11 0) (D 0) ($L11^H$ $L21^H$) if UPLO = 'L',

```
( L21  I ) (  0  A22 ) (  0       I    )
```

where the order of D is at most NB. The actual order is returned in the argument KB, and is either NB or NB-1, or N if N <= NB.

ZLAHEF_RK is an auxiliary routine called by ZHETRF_RK. It uses blocked code (calling Level 3 BLAS) to update the submatrix A11 (if UPLO = 'U') or A22 (if UPLO = 'L').

Syntax

Fortran specification:

```
use armpl_library

subroutine zlahef_rk(UPLO, N, NB, KB, A, LDA, E, IPIV, W, LDW, INFO)
```

C specification:

```
#include "armpl.h"

void zlahef_rk_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nb,
                armpl_int_t *kb, armpl_doublecomplex_t *a,
                const armpl_int_t *lda, armpl_doublecomplex_t *e,
                armpl_int_t *ipiv, armpl_doublecomplex_t *w,
                const armpl_int_t *ldw, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the Hermitian matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NB Input parameter.

NB is INTEGER

The maximum number of columns of the matrix A that should be factored. NB should be at least 2 to allow for 2-by-2 pivot blocks.

KB Output parameter.

KB is INTEGER

The number of columns of A that were actually factored. KB is either NB-1 or NB, or N if N <= NB.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U': the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced.

If UPLO = 'L': the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, contains: a) ONLY diagonal elements of the Hermitian block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Output parameter.

E is COMPLEX*16

E is an array, dimension (N). On exit, contains the superdiagonal (or subdiagonal) elements of the Hermitian block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) is set to 0; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) is set to 0.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is set to 0 in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

IPIV describes the permutation matrix P in the factorization of matrix A as follows. The absolute value of IPIV(k) represents the index of row and column that were interchanged with the k-th row and column. The value of UPLO describes the order in which the interchanges were applied. Also, the sign of IPIV represents the block structure of the Hermitian block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks which correspond to 1 or 2 interchanges at each factorization step.

If UPLO = 'U', (in factorization order, k decreases from N to 1): a) A single positive entry IPIV(k) > 0 means: D(k,k) is a 1-by-1 diagonal block. If IPIV(k) != k, rows and columns k and IPIV(k) were interchanged in the submatrix A(1:N,N-KB+1:N); If IPIV(k) = k, no interchange occurred.

b) A pair of consecutive negative entries IPIV(k) < 0 and IPIV(k-1) < 0 means: D(k-1:k,k-1:k) is a 2-by-2 diagonal block. (NOTE: negative entries in IPIV appear ONLY in pairs). 1) If -IPIV(k) != k, rows and columns k and -IPIV(k) were interchanged in the matrix A(1:N,N-KB+1:N). If -IPIV(k) = k, no interchange occurred. 2) If -IPIV(k-1) != k-1, rows and columns k-1 and -IPIV(k-1) were interchanged in the submatrix A(1:N,N-KB+1:N). If -IPIV(k-1) = k-1, no interchange occurred.

c) In both cases a) and b) is always ABS(IPIV(k)) <= k.

d) NOTE: Any entry IPIV(k) is always NONZERO on output.

If UPLO = 'L', (in factorization order, k increases from 1 to N): a) A single positive entry IPIV(k) > 0 means: D(k,k) is a 1-by-1 diagonal block. If IPIV(k) != k, rows and columns k and IPIV(k) were interchanged in the submatrix A(1:N,1:KB). If IPIV(k) = k, no interchange occurred.

b) A pair of consecutive negative entries IPIV(k) < 0 and IPIV(k+1) < 0 means: D(k:k+1,k:k+1) is a 2-by-2 diagonal block. (NOTE: negative entries in IPIV appear ONLY in pairs). 1) If -IPIV(k) != k, rows and columns k and -IPIV(k) were interchanged in the submatrix A(1:N,1:KB). If -IPIV(k) = k, no interchange occurred. 2) If -IPIV(k+1) != k+1, rows and columns k-1 and -IPIV(k-1) were interchanged in the submatrix A(1:N,1:KB). If -IPIV(k+1) = k+1, no interchange occurred.

c) In both cases a) and b) is always ABS(IPIV(k)) >= k.

d) NOTE: Any entry IPIV(k) is always NONZERO on output.

W Output parameter.

W is COMPLEX*16

W is an array, dimension (LDW, NB) .

LDW Input parameter.

LDW is INTEGER

The leading dimension of the array W. LDW >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit

< 0: If INFO = -k, the k-th argument had an illegal value

> 0: If INFO = k, the matrix A is singular, because: If UPLO = 'U': column k in the upper triangular part of A contains all zeros. If UPLO = 'L': column k in the lower triangular part of A contains all zeros.

Therefore D(k,k) is exactly zero, and superdiagonal elements of column k of U (or subdiagonal elements of column k of L) are all zeros. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

NOTE: INFO only stores the first occurrence of a singularity, any subsequent occurrence of singularity is not stored in INFO even though the factorization always completes.

Related Information

For this routine in other precisions, please see *clahef_rk*.

4.16.107 zlasyf_rk

ZLASYF_RK computes a partial factorization of a complex symmetric matrix A using the bounded Bunch-Kaufman (rook) diagonal pivoting method. The partial factorization has the form:

A = (I U12) (A11 0) (I 0) if UPLO = 'U', or:

```
( 0  U22 ) (  0   D  ) ( U12**T U22**T )
```

A = (L11 0) (D 0) ($L11^T$ $L21^T$) if UPLO = 'L',

```
( L21  I ) (  0  A22 ) (  0       I    )
```

where the order of D is at most NB. The actual order is returned in the argument KB, and is either NB or NB-1, or N if N <= NB.

ZLASYF_RK is an auxiliary routine called by ZSYTRF_RK. It uses blocked code (calling Level 3 BLAS) to update the submatrix A11 (if UPLO = 'U') or A22 (if UPLO = 'L').

Syntax

Fortran specification:

```
use armpl_library

subroutine zlasyf_rk(UPLO, N, NB, KB, A, LDA, E, IPIV, W, LDW, INFO)
```

C specification:

```
#include "armpl.h"

void zlasyf_rk_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nb,
                armpl_int_t *kb, armpl_doublecomplex_t *a,
                const armpl_int_t *lda, armpl_doublecomplex_t *e,
                armpl_int_t *ipiv, armpl_doublecomplex_t *w,
                const armpl_int_t *ldw, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NB Input parameter.

NB is INTEGER

The maximum number of columns of the matrix A that should be factored. NB should be at least 2 to allow for 2-by-2 pivot blocks.

KB Output parameter.

KB is INTEGER

The number of columns of A that were actually factored. KB is either NB-1 or NB, or N if N <= NB.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U': the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced.

If UPLO = 'L': the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, contains: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Output parameter.

E is COMPLEX*16

E is an array, dimension (N). On exit, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) is set to 0; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) is set to 0.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is set to 0 in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

IPIV describes the permutation matrix P in the factorization of matrix A as follows. The absolute value of IPIV(k) represents the index of row and column that were interchanged with the k-th row and column. The value of UPLO describes the order in which the interchanges were applied. Also, the sign of IPIV represents the block structure of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks which correspond to 1 or 2 interchanges at each factorization step.

If UPLO = 'U', (in factorization order, k decreases from N to 1): a) A single positive entry IPIV(k) > 0 means: D(k,k) is a 1-by-1 diagonal block. If IPIV(k) != k, rows and columns k and IPIV(k) were interchanged in the submatrix A(1:N,N-KB+1:N); If IPIV(k) = k, no interchange occurred.

b) A pair of consecutive negative entries IPIV(k) < 0 and IPIV(k-1) < 0 means: D(k-1:k,k-1:k) is a 2-by-2 diagonal block. (NOTE: negative entries in IPIV appear ONLY in pairs). 1) If -IPIV(k) != k, rows and columns k and -IPIV(k) were interchanged in the matrix A(1:N,N-KB+1:N). If -IPIV(k) = k, no interchange occurred. 2) If -IPIV(k-1) != k-1, rows and columns k-1 and -IPIV(k-1) were interchanged in the submatrix A(1:N,N-KB+1:N). If -IPIV(k-1) = k-1, no interchange occurred.

c) In both cases a) and b) is always ABS(IPIV(k)) <= k.

d) NOTE: Any entry IPIV(k) is always NONZERO on output.

If UPLO = 'L', (in factorization order, k increases from 1 to N): a) A single positive entry IPIV(k) > 0 means: D(k,k) is a 1-by-1 diagonal block. If IPIV(k) != k, rows and columns k and IPIV(k) were interchanged in the submatrix A(1:N,1:KB). If IPIV(k) = k, no interchange occurred.

b) A pair of consecutive negative entries IPIV(k) < 0 and IPIV(k+1) < 0 means: D(k:k+1,k:k+1) is a 2-by-2 diagonal block. (NOTE: negative entries in IPIV appear ONLY in pairs). 1) If -IPIV(k) != k, rows and columns k and -IPIV(k) were interchanged in the submatrix A(1:N,1:KB). If -IPIV(k) = k, no interchange occurred. 2) If -IPIV(k+1) != k+1, rows and columns k-1 and -IPIV(k-1) were interchanged in the submatrix A(1:N,1:KB). If -IPIV(k+1) = k+1, no interchange occurred.

c) In both cases a) and b) is always ABS(IPIV(k)) >= k.

d) NOTE: Any entry IPIV(k) is always NONZERO on output.

W Output parameter.

W is COMPLEX*16

W is an array, dimension (LDW, NB) .

LDW Input parameter.

LDW is INTEGER

The leading dimension of the array W. LDW >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit

< 0: If INFO = -k, the k-th argument had an illegal value

> 0: If INFO = k, the matrix A is singular, because: If UPLO = 'U': column k in the upper triangular part of A contains all zeros. If UPLO = 'L': column k in the lower triangular part of A contains all zeros.

Therefore D(k,k) is exactly zero, and superdiagonal elements of column k of U (or subdiagonal elements of column k of L) are all zeros. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

NOTE: INFO only stores the first occurrence of a singularity, any subsequent occurrence of singularity is not stored in INFO even though the factorization always completes.

Related Information

For this routine in other precisions, please see *clasyf_rk*, *dlasyf_rk* and *slasyf_rk*.

4.16.108 zsyconvf

If parameter WAY = 'C': ZSYCONVF converts the factorization output format used in ZSYTRF provided on entry in parameter A into the factorization output format used in ZSYTRF_RK (or ZSYTRF_BK) that is stored on exit in parameters A and E. It also coverts in place details of the intechanges stored in IPIV from the format used in ZSYTRF into the format used in ZSYTRF_RK (or ZSYTRF_BK).

If parameter WAY = 'R': ZSYCONVF performs the conversion in reverse direction, i.e. converts the factorization output format used in ZSYTRF_RK (or ZSYTRF_BK) provided on entry in parameters A and E into the factorization output format used in ZSYTRF that is stored on exit in parameter A. It also coverts in place details of the intechanges stored in IPIV from the format used in ZSYTRF_RK (or ZSYTRF_BK) into the format used in ZSYTRF.

ZSYCONVF can also convert in Hermitian matrix case, i.e. between formats used in ZHETRF and ZHETRF_RK (or ZHETRF_BK).

Syntax

Fortran specification:

```
use armpl_library

subroutine zsyconvf(UPLO, WAY, N, A, LDA, E, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void zsyconvf_(const char *uplo, const char *way, const armpl_int_t *n,
               armpl_doublecomplex_t *a, const armpl_int_t *lda,
               armpl_doublecomplex_t *e, armpl_int_t *ipiv, armpl_int_t *info,
               ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix A. = 'U': Upper triangular = 'L': Lower triangular

WAY Input parameter.

WAY is CHARACTER*1

= 'C': Convert = 'R': Revert

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). 1) If WAY ='C':

On entry, contains factorization details in format used in ZSYTRF: a) all elements of the symmetric block diagonal matrix D on the diagonal of A and on superdiagonal (or subdiagonal) of A, and b) If UPLO = 'U': multipliers used to obtain factor U in the superdiagonal part of A. If UPLO = 'L': multipliers used to obtain factor L in the superdiagonal part of A.

On exit, contains factorization details in format used in ZSYTRF_RK or ZSYTRF_BK: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

2) If WAY = 'R':

On entry, contains factorization details in format used in ZSYTRF_RK or ZSYTRF_BK: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

On exit, contains factorization details in format used in ZSYTRF: a) all elements of the symmetric block diagonal matrix D on the diagonal of A and on superdiagonal (or subdiagonal) of A, and b) If UPLO = 'U': multipliers used to obtain factor U in the superdiagonal part of A. If UPLO = 'L': multipliers used to obtain factor L in the superdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Input and output parameter.

E is COMPLEX*16

E is an array, dimension (N). 1) If WAY ='C':

On entry, just a workspace.

On exit, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) is set to 0; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) is set to 0.

2) If WAY = 'R':

On entry, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i),i=2:N, E(1) not referenced; If UPLO = 'L': E(i) = D(i+1,i),i=1:N-1, E(N) not referenced.

On exit, is not changed

IPIV Input and output parameter.

IPIV is INTEGER array, dimension (N)

1) If WAY ='C': On entry, details of the interchanges and the block structure of D in the format used in ZSYTRF. On exit, details of the interchanges and the block structure of D in the format used in ZSYTRF_RK (or ZSYTRF_BK).

1) If WAY ='R': On entry, details of the interchanges and the block structure of D in the format used in ZSYTRF_RK (or ZSYTRF_BK). On exit, details of the interchanges and the block structure of D in the format used in ZSYTRF.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csyconvf*, *dsyconvf* and *ssyconvf*.

4.16.109 zsyconvf_rook

If parameter WAY = 'C': ZSYCONVF_ROOK converts the factorization output format used in ZSYTRF_ROOK provided on entry in parameter A into the factorization output format used in ZSYTRF_RK (or ZSYTRF_BK) that is stored on exit in parameters A and E. IPIV format for ZSYTRF_ROOK and ZSYTRF_RK (or ZSYTRF_BK) is the same and is not converted.

If parameter WAY = 'R': ZSYCONVF_ROOK performs the conversion in reverse direction, i.e. converts the factorization output format used in ZSYTRF_RK (or ZSYTRF_BK) provided on entry in parameters A and E into the factorization output format used in ZSYTRF_ROOK that is stored on exit in parameter A. IPIV format for ZSYTRF_ROOK and ZSYTRF_RK (or ZSYTRF_BK) is the same and is not converted.

ZSYCONVF_ROOK can also convert in Hermitian matrix case, i.e. between formats used in ZHETRF_ROOK and ZHETRF_RK (or ZHETRF_BK).

Syntax

Fortran specification:

```
use armpl_library

subroutine zsyconvf_rook(UPLO, WAY, N, A, LDA, E, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void zsyconvf_rook_(const char *uplo, const char *way, const armpl_int_t *n,
                    armpl_doublecomplex_t *a, const armpl_int_t *lda,
                    armpl_doublecomplex_t *e, const armpl_int_t *ipiv,
                    armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix A. = 'U': Upper triangular = 'L': Lower triangular

WAY Input parameter.

WAY is CHARACTER*1

= 'C': Convert = 'R': Revert

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). 1) If WAY ='C':

On entry, contains factorization details in format used in ZSYTRF_ROOK: a) all elements of the symmetric block diagonal matrix D on the diagonal of A and on superdiagonal (or subdiagonal) of A, and b) If UPLO = 'U': multipliers used to obtain factor U in the superdiagonal part of A. If UPLO = 'L': multipliers used to obtain factor L in the superdiagonal part of A.

On exit, contains factorization details in format used in ZSYTRF_RK or ZSYTRF_BK: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

2) If WAY = 'R':

On entry, contains factorization details in format used in ZSYTRF_RK or ZSYTRF_BK: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

On exit, contains factorization details in format used in ZSYTRF_ROOK: a) all elements of the symmetric block diagonal matrix D on the diagonal of A and on superdiagonal (or subdiagonal) of A, and b) If UPLO = 'U': multipliers used to obtain factor U in the superdiagonal part of A. If UPLO = 'L': multipliers used to obtain factor L in the superdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Input and output parameter.

E is COMPLEX*16

E is an array, dimension (N). 1) If WAY ='C':

On entry, just a workspace.

On exit, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) is set to 0; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) is set to 0.

2) If WAY = 'R':

On entry, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i),i=2:N, E(1) not referenced; If UPLO = 'L': E(i) = D(i+1,i),i=1:N-1, E(N) not referenced.

On exit, is not changed

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

On entry, details of the interchanges and the block structure of D as determined: 1) by ZSYTRF_ROOK, if WAY ='C'; 2) by ZSYTRF_RK (or ZSYTRF_BK), if WAY ='R'. The IPIV format is the same for all these routines.

On exit, is not changed.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csyconvf_rook*, *dsyconvf_rook* and *ssyconvf_rook*.

4.16.110 zsysv_aa_2stage

ZSYSV_AA_2STAGE computes the solution to a complex system of linear equations

```
A * X = B,
```

where A is an N-by-N symmetric matrix and X and B are N-by-NRHS matrices.

Aasen's 2-stage algorithm is used to factor A as

```
A = U * T * U**H,  if UPLO = 'U', or
A = L * T * L**H,  if UPLO = 'L',
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and T is symmetric and band. The matrix T is then LU-factored with partial pivoting. The factored form of A is then used to solve the system of equations A * X = B.

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsysv_aa_2stage(UPLO, N, NRHS, A, LDA, TB, LTB, IPIV, IPIV2, B,
                           LDB, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zsysv_aa_2stage_(const char *uplo, const armpl_int_t *n,
                      const armpl_int_t *nrhs, armpl_doublecomplex_t *a,
                      const armpl_int_t *lda, armpl_doublecomplex_t *tb,
                      const armpl_int_t *ltb, armpl_int_t *ipiv,
                      armpl_int_t *ipiv2, armpl_doublecomplex_t *b,
                      const armpl_int_t *ldb, armpl_doublecomplex_t *work,
                      const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, L is stored below (or above) the subdiaonal blocks, when UPLO is 'L' (or 'U').

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TB Output parameter.

TB is COMPLEX*16

TB is an array, dimension (LTB). On exit, details of the LU factorization of the band matrix.

LTB Input parameter.

The size of the array TB. LTB >= 4*N, internally

used to select NB such that LTB >= (3*NB+1)*N.

If LTB = -1, then a workspace query is assumed; the routine only calculates the optimal size of LTB, returns this value as the first entry of TB, and no error message related to LTB is issued by XERBLA.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

On exit, it contains the details of the interchanges, i.e., the row and column k of A were interchanged with the row and column IPIV(k).

IPIV2 Output parameter.

IPIV is INTEGER array, dimension (N)

On exit, it contains the details of the interchanges, i.e., the row and column k of T were interchanged with the row and column IPIV(k).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

WORK Output parameter.

WORK is COMPLEX*16 workspace of size LWORK

LWORK Input parameter.

The size of WORK. LWORK >= N, internally used to select NB

such that LWORK >= N*NB.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, band LU factorization failed on i-th column

Related Information

For this routine in other precisions, please see *csysv_aa_2stage*, *dsysv_aa_2stage* and *ssysv_aa_2stage*. It also exists with a native C interface as *LAPACKE_zsysv_aa_2stage*.

4.16.111 zsytf2_rk

ZSYTF2_RK computes the factorization of a complex symmetric matrix A using the bounded Bunch-Kaufman (rook) diagonal pivoting method:

```
A = P*U*D*(U**T)*(P**T) or A = P*L*D*(L**T)*(P**T),
```

where U (or L) is unit upper (or lower) triangular matrix, U^T (or L^T) is the transpose of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the unblocked version of the algorithm, calling Level 2 BLAS. For more information see Further Details section.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsytf2_rk(UPLO, N, A, LDA, E, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void zsytf2_rk_(const char *uplo, const armpl_int_t *n,
                armpl_doublecomplex_t *a, const armpl_int_t *lda,
                armpl_doublecomplex_t *e, armpl_int_t *ipiv,
                armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U': the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced.

If UPLO = 'L': the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, contains: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D are stored on exit in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Output parameter.

E is COMPLEX*16

E is an array, dimension (N). On exit, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) is set to 0; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) is set to 0.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is set to 0 in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

IPIV describes the permutation matrix P in the factorization of matrix A as follows. The absolute value of IPIV(k) represents the index of row and column that were interchanged with the k-th row and column. The value of UPLO describes the order in which the interchanges were applied. Also, the sign of IPIV represents the block structure of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks which correspond to 1 or 2 interchanges at each factorization step. For more info see Further Details section.

If UPLO = 'U', (in factorization order, k decreases from N to 1): a) A single positive entry IPIV(k) > 0 means: D(k,k) is a 1-by-1 diagonal block. If IPIV(k) != k, rows and columns k and IPIV(k) were interchanged in the matrix A(1:N,1:N); If IPIV(k) = k, no interchange occurred.

b) A pair of consecutive negative entries IPIV(k) < 0 and IPIV(k-1) < 0 means: D(k-1:k,k-1:k) is a 2-by-2 diagonal block. (NOTE: negative entries in IPIV appear ONLY in pairs). 1) If -IPIV(k) != k, rows and columns k and -IPIV(k) were interchanged in the matrix A(1:N,1:N). If -IPIV(k) = k, no interchange occurred. 2) If -IPIV(k-1) != k-1, rows and columns k-1 and -IPIV(k-1) were interchanged in the matrix A(1:N,1:N). If -IPIV(k-1) = k-1, no interchange occurred.

c) In both cases a) and b), always ABS(IPIV(k)) <= k.

d) NOTE: Any entry IPIV(k) is always NONZERO on output.

If UPLO = 'L', (in factorization order, k increases from 1 to N): a) A single positive entry IPIV(k) > 0 means: D(k,k) is a 1-by-1 diagonal block. If IPIV(k) != k, rows and columns k and IPIV(k) were interchanged in the matrix A(1:N,1:N). If IPIV(k) = k, no interchange occurred.

b) A pair of consecutive negative entries IPIV(k) < 0 and IPIV(k+1) < 0 means: D(k:k+1,k:k+1) is a 2-by-2 diagonal block. (NOTE: negative entries in IPIV appear ONLY in pairs). 1) If -IPIV(k) != k, rows and columns k and -IPIV(k) were interchanged in the matrix A(1:N,1:N). If -IPIV(k) = k, no interchange occurred. 2) If -IPIV(k+1) != k+1, rows and columns k-1 and -IPIV(k-1) were interchanged in the matrix A(1:N,1:N). If -IPIV(k+1) = k+1, no interchange occurred.

c) In both cases a) and b), always ABS(IPIV(k)) >= k.

d) NOTE: Any entry IPIV(k) is always NONZERO on output.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit

< 0: If INFO = -k, the k-th argument had an illegal value

> 0: If INFO = k, the matrix A is singular, because: If UPLO = 'U': column k in the upper triangular part of A contains all zeros. If UPLO = 'L': column k in the lower triangular part of A contains all zeros.

Therefore D(k,k) is exactly zero, and superdiagonal elements of column k of U (or subdiagonal elements of column k of L) are all zeros. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

NOTE: INFO only stores the first occurrence of a singularity, any subsequent occurrence of singularity is not stored in INFO even though the factorization always completes.

Related Information

For this routine in other precisions, please see *csytf2_rk*, *dsytf2_rk* and *ssytf2_rk*.

4.16.112 zsytrf_aa_2stage

ZSYTRF_AA_2STAGE computes the factorization of a complex symmetric matrix A using the Aasen's algorithm. The form of the factorization is

```
A = U*T*U**T  or  A = L*T*L**T
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and T is a complex symmetric band matrix with the bandwidth of NB (NB is internally selected and stored in TB(1), and T is LU factorized with partial pivoting).

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsytrf_aa_2stage(UPLO, N, A, LDA, TB, LTB, IPIV, IPIV2, WORK,
                            LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zsytrf_aa_2stage_(const char *uplo, const armpl_int_t *n,
                       armpl_doublecomplex_t *a, const armpl_int_t *lda,
                       armpl_doublecomplex_t *tb, const armpl_int_t *ltb,
                       armpl_int_t *ipiv, armpl_int_t *ipiv2,
                       armpl_doublecomplex_t *work, const armpl_int_t *lwork,
                       armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, L is stored below (or above) the subdiaonal blocks, when UPLO is 'L' (or 'U').

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TB Output parameter.

TB is COMPLEX*16

TB is an array, dimension (LTB). On exit, details of the LU factorization of the band matrix.

LTB Input parameter.

The size of the array TB. LTB >= 4*N, internally

used to select NB such that LTB >= (3*NB+1)*N.

If LTB = -1, then a workspace query is assumed; the routine only calculates the optimal size of LTB, returns this value as the first entry of TB, and no error message related to LTB is issued by XERBLA.

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

On exit, it contains the details of the interchanges, i.e., the row and column k of A were interchanged with the row and column IPIV(k).

IPIV2 Output parameter.

IPIV is INTEGER array, dimension (N)

On exit, it contains the details of the interchanges, i.e., the row and column k of T were interchanged with the row and column IPIV(k).

WORK Output parameter.

WORK is COMPLEX*16 workspace of size LWORK

LWORK Input parameter.

The size of WORK. LWORK >= N, internally used to select NB

such that LWORK >= N*NB.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = i, band LU factorization failed on i-th column

Related Information

For this routine in other precisions, please see *csytrf_aa_2stage*, *dsytrf_aa_2stage* and *ssytrf_aa_2stage*. It also exists with a native C interface as *LAPACKE_zsytrf_aa_2stage*.

4.16.113 zsytri_3x

ZSYTRI_3X computes the inverse of a complex symmetric indefinite matrix A using the factorization computed by ZSYTRF_RK or ZSYTRF_BK:

```
A = P*U*D*(U**T)*(P**T) or A = P*L*D*(L**T)*(P**T),
```

where U (or L) is unit upper (or lower) triangular matrix, U^T (or L^T) is the transpose of U (or L), P is a permutation matrix, P^T is the transpose of P, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the blocked version of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsytri_3x(UPLO, N, A, LDA, E, IPIV, WORK, NB, INFO)
```

C specification:

```
#include "armpl.h"

void zsytri_3x_(const char *uplo, const armpl_int_t *n,
                armpl_doublecomplex_t *a, const armpl_int_t *lda,
                const armpl_doublecomplex_t *e, const armpl_int_t *ipiv,
                armpl_doublecomplex_t *work, const armpl_int_t *nb,
                armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, diagonal of the block diagonal matrix D and factors U or L as computed by ZSYTRF_RK and ZSYTRF_BK: a) ONLY diagonal elements of the symmetric block diagonal matrix D on the diagonal of A, i.e. D(k,k) = A(k,k); (superdiagonal (or subdiagonal) elements of D should be provided on entry in array E), and b) If UPLO = 'U': factor U in the superdiagonal part of A. If UPLO = 'L': factor L in the subdiagonal part of A.

On exit, if INFO = 0, the symmetric inverse of the original matrix. If UPLO = 'U': the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; If UPLO = 'L': the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Input parameter.

E is COMPLEX*16

E is an array, dimension (N). On entry, contains the superdiagonal (or subdiagonal) elements of the symmetric block diagonal matrix D with 1-by-1 or 2-by-2 diagonal blocks, where If UPLO = 'U': E(i) = D(i-1,i), i=2:N, E(1) not referenced; If UPLO = 'L': E(i) = D(i+1,i), i=1:N-1, E(N) not referenced.

NOTE: For 1-by-1 diagonal block D(k), where 1 <= k <= N, the element E(k) is not referenced in both UPLO = 'U' or UPLO = 'L' cases.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZSYTRF_RK or ZSYTRF_BK.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N+NB+1,NB+3). .

NB Input parameter.

NB is INTEGER

Block size.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

Related Information

For this routine in other precisions, please see *csytri_3x*, *dsytri_3x* and *ssytri_3x*.

4.16.114 zsytrs_aa_2stage

ZSYTRS_AA_2STAGE solves a system of linear equations A*X = B with a complex symmetric matrix A using the factorization A = U*T*U^T or A = L*T*L^T computed by ZSYTRF_AA_2STAGE.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsytrs_aa_2stage(UPLO, N, NRHS, A, LDA, TB, LTB, IPIV, IPIV2, B,
                            LDB, INFO)
```

C specification:

```
#include "armpl.h"

void zsytrs_aa_2stage_(const char *uplo, const armpl_int_t *n,
                       const armpl_int_t *nrhs,
                       const armpl_doublecomplex_t *a, const armpl_int_t *lda,
                       armpl_doublecomplex_t *tb, const armpl_int_t *ltb,
                       const armpl_int_t *ipiv, const armpl_int_t *ipiv2,
                       armpl_doublecomplex_t *b, const armpl_int_t *ldb,
                       armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*T*U^T ; = 'L': Lower triangular, form is A = L*T*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). Details of factors computed by ZSYTRF_AA_2STAGE.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TB Output parameter.

TB is COMPLEX*16

TB is an array, dimension (LTB). Details of factors computed by ZSYTRF_AA_2STAGE.

LTB Input parameter.

The size of the array TB. LTB >= 4*N.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges as computed by ZSYTRF_AA_2STAGE.

IPIV2 Input parameter.

IPIV2 is INTEGER array, dimension (N)

Details of the interchanges as computed by ZSYTRF_AA_2STAGE.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csytrs_aa_2stage*, *dsytrs_aa_2stage* and *ssytrs_aa_2stage*. It also exists with a native C interface as *LAPACKE_zsytrs_aa_2stage*.

4.16.115 ztrevc3

`ztrevc3` computes some or all of the right and/or left eigenvectors of a complex upper triangular matrix T. Matrices of this type are produced by the Schur factorization of a complex general matrix: A = Q*T*Q^H , as computed by ZHSEQR.

The right eigenvector x and the left eigenvector y of T corresponding to an eigenvalue w are defined by:

```
T*x = w*x,      (y**H)*T = w*(y**H)
```

where y^H denotes the conjugate transpose of the vector y. The eigenvalues are not input to this routine, but are read directly from the diagonal of T.

This routine returns the matrices X and/or Y of right and left eigenvectors of T, or the products Q*X and/or Q*Y, where Q is an input matrix. If Q is the unitary factor that reduces a matrix A to Schur form T, then Q*X and Q*Y are the matrices of right and left eigenvectors of A.

This uses a Level 3 BLAS version of the back transformation.

Syntax

Fortran specification:

```
use armpl_library

subroutine ztrevc3(SIDE, HOWMNY, SELECT, N, T, LDT, VL, LDVL, VR, LDVR, MM, M,
                   WORK, LWORK, RWORK, LRWORK, INFO)
```

C specification:

```
#include "armpl.h"

void ztrevc3_(const char *side, const char *howmny, const armpl_int_t *select,
              const armpl_int_t *n, armpl_doublecomplex_t *t,
              const armpl_int_t *ldt, armpl_doublecomplex_t *vl,
              const armpl_int_t *ldvl, armpl_doublecomplex_t *vr,
              const armpl_int_t *ldvr, const armpl_int_t *mm, armpl_int_t *m,
              armpl_doublecomplex_t *work, const armpl_int_t *lwork,
              double *rwork, const armpl_int_t *lrwork, armpl_int_t *info,
              ... );
```

Parameters

SIDE Input parameter.

`SIDE` is `CHARACTER*1`

= 'R': compute right eigenvectors only; = 'L': compute left eigenvectors only; = 'B': compute both right and left eigenvectors.

HOWMNY Input parameter.

`HOWMNY` is `CHARACTER*1`

= 'A': compute all right and/or left eigenvectors; = 'B': compute all right and/or left eigenvectors, backtransformed using the matrices supplied in VR and/or VL; = 'S': compute selected right and/or left eigenvectors, as indicated by the logical array SELECT.

SELECT Input parameter.

SELECT is LOGICAL

SELECT is an array, dimension (N). If HOWMNY = 'S', SELECT specifies the eigenvectors to be computed. The eigenvector corresponding to the j-th eigenvalue is computed if SELECT(j) = .TRUE.. Not referenced if HOWMNY = 'A' or 'B'.

N Input parameter.

N is INTEGER

The order of the matrix T. N >= 0.

T Input and output parameter.

T is COMPLEX*16

T is an array, dimension (LDT, N). The upper triangular matrix T. T is modified, but restored on exit.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

VL Input and output parameter.

VL is COMPLEX*16

VL is an array, dimension (LDVL, MM). On entry, if SIDE = 'L' or 'B' and HOWMNY = 'B', VL must contain an N-by-N matrix Q (usually the unitary matrix Q of Schur vectors returned by ZHSEQR). On exit, if SIDE = 'L' or 'B', VL contains: if HOWMNY = 'A', the matrix Y of left eigenvectors of T; if HOWMNY = 'B', the matrix Q*Y; if HOWMNY = 'S', the left eigenvectors of T specified by SELECT, stored consecutively in the columns of VL, in the same order as their eigenvalues. Not referenced if SIDE = 'R'.

LDVL Input parameter.

LDVL is INTEGER

The leading dimension of the array VL. LDVL >= 1, and if SIDE = 'L' or 'B', LDVL >= N.

VR Input and output parameter.

VR is COMPLEX*16

VR is an array, dimension (LDVR, MM). On entry, if SIDE = 'R' or 'B' and HOWMNY = 'B', VR must contain an N-by-N matrix Q (usually the unitary matrix Q of Schur vectors returned by ZHSEQR). On exit, if SIDE = 'R' or 'B', VR contains: if HOWMNY = 'A', the matrix X of right eigenvectors of T; if HOWMNY = 'B', the matrix Q*X; if HOWMNY = 'S', the right eigenvectors of T specified by SELECT, stored consecutively in the columns of VR, in the same order as their eigenvalues. Not referenced if SIDE = 'L'.

LDVR Input parameter.

LDVR is INTEGER

The leading dimension of the array VR. LDVR >= 1, and if SIDE = 'R' or 'B', LDVR >= N.

MM Input parameter.

MM is INTEGER

The number of columns in the arrays VL and/or VR. MM >= M.

M Output parameter.

M is INTEGER

The number of columns in the arrays VL and/or VR actually used to store the eigenvectors. If HOWMNY = 'A' or 'B', M is set to N. Each selected eigenvector occupies one column.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (MAX(1, LWORK)) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of array WORK. LWORK >= max(1,2*N). For optimum performance, LWORK >= N + 2*N*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (LRWORK) .

LRWORK Input parameter.

LRWORK is INTEGER

The dimension of array RWORK. LRWORK >= max(1, N).

If LRWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the RWORK array, returns this value as the first entry of the RWORK array, and no error message related to LRWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctrevc3*, *dtrevc3* and *strevc3*.

4.16.116 ztrtrs

ztrtrs solves a triangular system of the form

```
A * X = B,  A**T * X = B,  or  A**H * X = B,
```

where A is a triangular matrix of order N, and B is an N-by-NRHS matrix. A check is made to verify that A is nonsingular.

Syntax

Fortran specification:

```
use armpl_library

subroutine ztrtrs(UPLO, TRANS, DIAG, N, NRHS, A, LDA, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void ztrtrs_(const char *uplo, const char *trans, const char *diag,
             const armpl_int_t *n, const armpl_int_t *nrhs,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_doublecomplex_t *b, const armpl_int_t *ldb,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate transpose)

DIAG Input parameter.

DIAG is CHARACTER*1

= 'N': A is non-unit triangular; = 'U': A is unit triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the right hand side matrix B. On exit, if INFO = 0, the solution matrix X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = i, the i-th diagonal element of A is zero, indicating that the matrix is singular and the solutions X have not been computed.

Related Information

For this routine in other precisions, please see *ctrtrs*, *dtrtrs* and *strtrs*. It also exists with a native C interface as *LAPACKE_ztrtrs*.

4.17 LAPACK auxiliary routines

4.17.1 cgbtf2

cgbtf2 computes an LU factorization of a complex m-by-n band matrix A using partial pivoting with row interchanges.

This is the unblocked version of the algorithm, calling Level 2 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgbtf2(M, N, KL, KU, AB, LDAB, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void cgbtf2_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *kl, const armpl_int_t *ku,
             armpl_singlecomplex_t *ab, const armpl_int_t *ldab,
             armpl_int_t *ipiv, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

AB Input and output parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows KL+1 to 2*KL+KU+1; rows 1 to KL of the array need not be set. The j-th column of A is stored in the j-th column of the array AB as follows: AB(kl+ku+1+i-j,j) = A(i,j) for max(1,j-ku)<=i<=min(m,j+kl)

On exit, details of the factorization: U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1. See below for further details.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= 2*KL+KU+1.

IPIV Output parameter.

IPIV is INTEGER array, dimension (min(M, N))

The pivot indices; for 1 <= i <= min(M, N), row i of the matrix was interchanged with row IPIV(i).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = +i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *dgbtf2*, *sgbtf2* and *zgbtf2*.

4.17.2 cgebd2

cgebd2 reduces a complex general m by n matrix A to upper or lower real bidiagonal form B by a unitary transformation: $Q^H * A * P = B$.

If m >= n, B is upper bidiagonal; if m < n, B is lower bidiagonal.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgebd2(M, N, A, LDA, D, E, TAUQ, TAUP, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgebd2_(const armpl_int_t *m, const armpl_int_t *n,
             armpl_singlecomplex_t *a, const armpl_int_t *lda, float *d,
             float *e, armpl_singlecomplex_t *tauq,
```

(continues on next page)

(continued from previous page)

```
            armpl_singlecomplex_t *taup, armpl_singlecomplex_t *work,
            armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows in the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns in the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the m by n general matrix to be reduced. On exit, if m >= n, the diagonal and the first superdiagonal are overwritten with the upper bidiagonal matrix B; the elements below the diagonal, with the array TAUQ, represent the unitary matrix Q as a product of elementary reflectors, and the elements above the first superdiagonal, with the array TAUP, represent the unitary matrix P as a product of elementary reflectors; if m < n, the diagonal and the first subdiagonal are overwritten with the lower bidiagonal matrix B; the elements below the first subdiagonal, with the array TAUQ, represent the unitary matrix Q as a product of elementary reflectors, and the elements above the diagonal, with the array TAUP, represent the unitary matrix P as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

D Output parameter.

D is REAL

D is an array, dimension (min(M, N)). The diagonal elements of the bidiagonal matrix B: D(i) = A(i,i).

E Output parameter.

E is REAL

E is an array, dimension (min(M, N)-1). The off-diagonal elements of the bidiagonal matrix B: if m >= n, E(i) = A(i,i+1) for i = 1,2,...,n-1; if m < n, E(i) = A(i+1,i) for i = 1,2,...,m-1.

TAUQ Output parameter.

TAUQ is COMPLEX

TAUQ is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors which represent the unitary matrix Q. See Further Details.

TAUP Output parameter.

TAUP is COMPLEX

TAUP is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors which represent the unitary matrix P. See Further Details.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (max(M, N)) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *dgebd2*, *sgebd2* and *zgebd2*.

4.17.3 cgehd2

cgehd2 reduces a complex general matrix A to upper Hessenberg form H by a unitary similarity transformation: $Q^H * A * Q = H$.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgehd2(N, ILO, IHI, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgehd2_(const armpl_int_t *n, const armpl_int_t *ilo,
             const armpl_int_t *ihi, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_singlecomplex_t *tau,
             armpl_singlecomplex_t *work, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

It is assumed that A is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N. ILO and IHI are normally set by a previous call to CGEBAL; otherwise they should be set to 1 and N respectively. See Further Details. 1 <= ILO <= IHI <= max(1, N).

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the n by n general matrix to be reduced. On exit, the upper triangle and the first subdiagonal of A are overwritten with the upper Hessenberg matrix H, and the elements below the first subdiagonal, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TAU Output parameter.

TAU is COMPLEX

TAU is an array, dimension (N-1). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *dgehd2*, *sgehd2* and *zgehd2*.

4.17.4 cgelq2

cgelq2 computes an LQ factorization of a complex m by n matrix A: A = L * Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgelq2(M, N, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgelq2_(const armpl_int_t *m, const armpl_int_t *n,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             armpl_singlecomplex_t *tau, armpl_singlecomplex_t *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the m by n matrix A. On exit, the elements on and below the diagonal of the array contain the m by min(m,n) lower trapezoidal matrix L (L is lower triangular if m <= n); the elements above the diagonal, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is COMPLEX

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (M) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dgelq2*, *sgelq2* and *zgelq2*. It also exists with a native C interface as *LAPACKE_cgelq2*.

4.17.5 cgelqt3

cgelqt3 recursively computes a LQ factorization of a complex M-by-N matrix A, using the compact WY representation of
Based on the algorithm of Elmroth and Gustavson, IBM J. Res. Develop. Vol 44 No. 4 July 2000.

Syntax

Fortran specification:

```
use armpl_library

recursive subroutine cgelqt3(M, N, A, LDA, T, LDT, INFO)
```

C specification:

```
#include "armpl.h"

void cgelqt3_(const armpl_int_t *m, const armpl_int_t *n,
              armpl_singlecomplex_t *a, const armpl_int_t *lda,
              armpl_singlecomplex_t *t, const armpl_int_t *ldt,
              armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M =< N.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the real M-by-N matrix A. On exit, the elements on and below the diagonal contain the N-by-N lower triangular matrix L; the elements above the diagonal are the rows of V. See below for further details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

T Output parameter.

T is COMPLEX

T is an array, dimension (LDT, N). The N-by-N upper triangular factor of the block reflector. The elements on and above the diagonal contain the block reflector T; the elements below the diagonal are not used. See below for further details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dgelqt3*, *sgelqt3* and *zgelqt3*.

4.17.6 cgeql2

cgeql2 computes a QL factorization of a complex m by n matrix A: A = Q * L.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgeql2(M, N, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgeql2_(const armpl_int_t *m, const armpl_int_t *n,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             armpl_singlecomplex_t *tau, armpl_singlecomplex_t *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the m by n matrix A. On exit, if m >= n, the lower triangle of the subarray A(m-n+1:m,1:n) contains the n by n lower triangular matrix L; if m <= n, the elements on and below the (n-m)-th superdiagonal contain the m by n lower trapezoidal matrix L; the remaining elements, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is COMPLEX

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dgeql2*, *sgeql2* and *zgeql2*.

4.17.7 cgeqr2

cgeqr2 computes a QR factorization of a complex m by n matrix A: A = Q * R.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgeqr2(M, N, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgeqr2_(const armpl_int_t *m, const armpl_int_t *n,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             armpl_singlecomplex_t *tau, armpl_singlecomplex_t *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the m by n matrix A. On exit, the elements on and above the diagonal of the array contain the min(m,n) by n upper trapezoidal matrix R (R is upper triangular if m >= n); the elements below the diagonal, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is COMPLEX

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dgeqr2*, *sgeqr2* and *zgeqr2*. It also exists with a native C interface as *LAPACKE_cgeqr2*.

4.17.8 cgeqr2p

cgeqr2p computes a QR factorization of a complex m by n matrix A: A = Q * R. The diagonal entries of R are real and nonnegative.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgeqr2p(M, N, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgeqr2p_(const armpl_int_t *m, const armpl_int_t *n,
              armpl_singlecomplex_t *a, const armpl_int_t *lda,
              armpl_singlecomplex_t *tau, armpl_singlecomplex_t *work,
              armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the m by n matrix A. On exit, the elements on and above the diagonal of the array contain the min(m,n) by n upper trapezoidal matrix R (R is upper triangular if m >= n). The diagonal entries of R are real and nonnegative; the elements below the diagonal, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is COMPLEX

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dgeqr2p*, *sgeqr2p* and *zgeqr2p*.

4.17.9 cgeqrt2

cgeqrt2 computes a QR factorization of a complex M-by-N matrix A, using the compact WY representation of Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgeqrt2(M, N, A, LDA, T, LDT, INFO)
```

C specification:

```
#include "armpl.h"

void cgeqrt2_(const armpl_int_t *m, const armpl_int_t *n,
              armpl_singlecomplex_t *a, const armpl_int_t *lda,
              armpl_singlecomplex_t *t, const armpl_int_t *ldt,
              armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= N.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the complex M-by-N matrix A. On exit, the elements on and above the diagonal contain the N-by-N upper triangular matrix R; the elements below the diagonal are the columns of V. See below for further details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

T Output parameter.

T is COMPLEX

T is an array, dimension (LDT, N). The N-by-N upper triangular factor of the block reflector. The elements on and above the diagonal contain the block reflector T; the elements below the diagonal are not used. See below for further details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dgeqrt2*, *sgeqrt2* and *zgeqrt2*. It also exists with a native C interface as *LAPACKE_cgeqrt2*.

4.17.10 cgeqrt3

cgeqrt3 recursively computes a QR factorization of a complex M-by-N matrix A, using the compact WY representation of Q.

Based on the algorithm of Elmroth and Gustavson, IBM J. Res. Develop. Vol 44 No. 4 July 2000.

Syntax

Fortran specification:

```
use armpl_library

recursive subroutine cgeqrt3(M, N, A, LDA, T, LDT, INFO)
```

C specification:

```
#include "armpl.h"

void cgeqrt3_(const armpl_int_t *m, const armpl_int_t *n,
              armpl_singlecomplex_t *a, const armpl_int_t *lda,
              armpl_singlecomplex_t *t, const armpl_int_t *ldt,
              armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= N.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the complex M-by-N matrix A. On exit, the elements on and above the diagonal contain the N-by-N upper triangular matrix R; the elements below the diagonal are the columns of V. See below for further details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

T Output parameter.

T is COMPLEX

T is an array, dimension (LDT, N). The N-by-N upper triangular factor of the block reflector. The elements on and above the diagonal contain the block reflector T; the elements below the diagonal are not used. See below for further details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dgeqrt3*, *sgeqrt3* and *zgeqrt3*. It also exists with a native C interface as *LAPACKE_cgeqrt3*.

4.17.11 cgerq2

cgerq2 computes an RQ factorization of a complex m by n matrix A: A = R * Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgerq2(M, N, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgerq2_(const armpl_int_t *m, const armpl_int_t *n,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             armpl_singlecomplex_t *tau, armpl_singlecomplex_t *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the m by n matrix A. On exit, if m <= n, the upper triangle of the subarray A(1:m,n-m+1:n) contains the m by m upper triangular matrix R; if m >= n, the elements on and above the (m-n)-th subdiagonal contain the m by n upper trapezoidal matrix R; the remaining elements, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is COMPLEX

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (M) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dgerq2*, *sgerq2* and *zgerq2*.

4.17.12 cgesc2

cgesc2 solves a system of linear equations

```
A * X = scale* RHS
```

with a general N-by-N matrix A using the LU factorization with complete pivoting computed by CGETC2.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgesc2(N, A, LDA, RHS, IPIV, JPIV, SCALE)
```

C specification:

```
#include "armpl.h"

void cgesc2_(const armpl_int_t *n, const armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_singlecomplex_t *rhs,
             const armpl_int_t *ipiv, const armpl_int_t *jpiv, float *scale);
```

Parameters

N Input parameter.

N is INTEGER

The number of columns of the matrix A.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the LU part of the factorization of the n-by-n matrix A computed by CGETC2: A = P * L * U * Q

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

RHS Input and output parameter.

RHS is COMPLEX

RHS is an array, dimension N.. On entry, the right hand side vector b. On exit, the solution vector X.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N).

The pivot indices; for 1 <= i <= N, row i of the matrix has been interchanged with row IPIV(i).

JPIV Input parameter.

JPIV is INTEGER array, dimension (N).

The pivot indices; for 1 <= j <= N, column j of the matrix has been interchanged with column JPIV(j).

SCALE Output parameter.

SCALE is REAL

On exit, SCALE contains the scale factor. SCALE is chosen 0 <= SCALE <= 1 to prevent owerflow in the solution.

Related Information

For this routine in other precisions, please see *dgesc2*, *sgesc2* and *zgesc2*.

4.17.13 cgetc2

cgetc2 computes an LU factorization, using complete pivoting, of the n-by-n matrix A. The factorization has the form A = P * L * U * Q, where P and Q are permutation matrices, L is lower triangular with unit diagonal elements and U is upper triangular.

This is a level 1 BLAS version of the algorithm.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgetc2(N, A, LDA, IPIV, JPIV, INFO)
```

C specification:

```
#include "armpl.h"

void cgetc2_(const armpl_int_t *n, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *ipiv, armpl_int_t *jpiv,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the n-by-n matrix to be factored. On exit, the factors L and U from the factorization A = P*L*U*Q; the unit diagonal elements of L are not stored. If U(k, k) appears to be less than SMIN, U(k, k) is given the value of SMIN, giving a nonsingular perturbed system.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N).

The pivot indices; for 1 <= i <= N, row i of the matrix has been interchanged with row IPIV(i).

JPIV Output parameter.

JPIV is INTEGER array, dimension (N).

The pivot indices; for 1 <= j <= N, column j of the matrix has been interchanged with column JPIV(j).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit > 0: if INFO = k, U(k, k) is likely to produce overflow if one tries to solve for x in Ax = b. So U is perturbed to avoid the overflow.

Related Information

For this routine in other precisions, please see *dgetc2*, *sgetc2* and *zgetc2*.

4.17.14 cgetf2

`cgetf2` computes an LU factorization of a general m-by-n matrix A using partial pivoting with row interchanges.

The factorization has the form

```
A = P * L * U
```

where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if m > n), and U is upper triangular (upper trapezoidal if m < n).

This is the right-looking Level 2 BLAS version of the algorithm.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgetf2(M, N, A, LDA, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void cgetf2_(const armpl_int_t *m, const armpl_int_t *n,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             armpl_int_t *ipiv, armpl_int_t *info);
```

Parameters

M Input parameter.

`M` is `INTEGER`

The number of rows of the matrix `A`. `M` >= 0.

N Input parameter.

`N` is `INTEGER`

The number of columns of the matrix `A`. `N` >= 0.

A Input and output parameter.

`A` is `COMPLEX`

`A` is an array, dimension (`LDA`, `N`). On entry, the m by n matrix to be factored. On exit, the factors L and U from the factorization `A` = P*L*U; the unit diagonal elements of L are not stored.

LDA Input parameter.

`LDA` is `INTEGER`

The leading dimension of the array `A`. `LDA` >= max(1, `M`).

IPIV Output parameter.

`IPIV` is `INTEGER` array, dimension (min(`M`, `N`))

The pivot indices; for 1 <= i <= min(`M`, `N`), row i of the matrix was interchanged with row IPIV(i).

INFO Output parameter.

`INFO` is `INTEGER`

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value > 0: if INFO = k, U(k,k) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *dgetf2*, *sgetf2* and *zgetf2*. It also exists with a native C interface as *LAPACKE_cgetf2*.

4.17.15 cgsvj0

cgsvj0 is called from CGESVJ as a pre-processor and that is its main purpose. It applies Jacobi rotations in the same way as CGESVJ does, but it does not check convergence (stopping criterion). Few tuning parameters (marked by [TP]) are available for the implementer.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgsvj0(JOBV, M, N, A, LDA, D, SVA, MV, V, LDV, EPS, SFMIN, TOL,
                  NSWEEP, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgsvj0_(const char *jobv, const armpl_int_t *m, const armpl_int_t *n,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             armpl_singlecomplex_t *d, float *sva, const armpl_int_t *mv,
             armpl_singlecomplex_t *v, const armpl_int_t *ldv,
             const float *eps, const float *sfmin, const float *tol,
             const armpl_int_t *nsweep, armpl_singlecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

JOBV Input parameter.

JOBV is CHARACTER*1

Specifies whether the output from this procedure is used to compute the matrix V: = 'V': the product of the Jacobi rotations is accumulated by postmulyiplying the N-by-N array V. (See the description of V.) = 'A': the product of the Jacobi rotations is accumulated by postmulyiplying the MV-by-N array V. (See the descriptions of MV and V.) = 'N': the Jacobi rotations are not accumulated.

M Input parameter.

M is INTEGER

The number of rows of the input matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the input matrix A. M >= N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, M-by-N matrix A, such that A*diag(D) represents the input matrix. On exit, A_onexit * diag(D_onexit) represents the input matrix A*diag(D) post-multiplied by a sequence of Jacobi rotations, where the rotation threshold and the total number of sweeps are given in TOL and NSWEEP, respectively. (See the descriptions of D, TOL and NSWEEP.)

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

D Input and output parameter.

D is COMPLEX

D is an array, dimension (N). The array D accumulates the scaling factors from the complex scaled Jacobi rotations. On entry, A*diag(D) represents the input matrix. On exit, A_onexit*diag(D_onexit) represents the input matrix post-multiplied by a sequence of Jacobi rotations, where the rotation threshold and the total number of sweeps are given in TOL and NSWEEP, respectively. (See the descriptions of A, TOL and NSWEEP.)

SVA Input and output parameter.

SVA is REAL

SVA is an array, dimension (N). On entry, SVA contains the Euclidean norms of the columns of the matrix A*diag(D). On exit, SVA contains the Euclidean norms of the columns of the matrix A_onexit*diag(D_onexit).

MV Input parameter.

MV is INTEGER

If JOBV .EQ. 'A', then MV rows of V are post-multipled by a sequence of Jacobi rotations. If JOBV = 'N', then MV is not referenced.

V Input and output parameter.

V is COMPLEX

V is an array, dimension (LDV, N). If JOBV .EQ. 'V' then N rows of V are post-multipled by a sequence of Jacobi rotations. If JOBV .EQ. 'A' then MV rows of V are post-multipled by a sequence of Jacobi rotations. If JOBV = 'N', then V is not referenced.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V, LDV >= 1. If JOBV = 'V', LDV .GE. N. If JOBV = 'A', LDV .GE. MV.

EPS Input parameter.

EPS is REAL

EPS = SLAMCH('Epsilon')

SFMIN Input parameter.

SFMIN is REAL

SFMIN = SLAMCH('Safe Minimum')

TOL Input parameter.

TOL is REAL

TOL is the threshold for Jacobi rotations. For a pair A(:,p), A(:,q) of pivot columns, the Jacobi rotation is applied only if ABS(COS(angle(A(:,p),A(:,q)))) .GT. TOL.

NSWEEP Input parameter.

NSWEEP is INTEGER

NSWEEP is the number of sweeps of Jacobi rotations to be performed.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

LWORK is the dimension of WORK. LWORK .GE. M.

INFO Output parameter.

INFO is INTEGER

= 0 : successful exit. < 0 : if INFO = -i, then the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dgsvj0*, *sgsvj0* and *zgsvj0*.

4.17.16 cgsvj1

cgsvj1 is called from CGESVJ as a pre-processor and that is its main purpose. It applies Jacobi rotations in the same way as CGESVJ does, but it targets only particular pivots and it does not check convergence (stopping criterion). Few tunning parameters (marked by [TP]) are available for the implementer.

Further Details

cgsvj1 applies few sweeps of Jacobi rotations in the column space of the input M-by-N matrix A. The pivot pairs are taken from the (1,2) off-diagonal block in the corresponding N-by-N Gram matrix A^T * A. The block-entries (tiles) of the (1,2) off-diagonal block are marked by the [x]'s in the following scheme:

```
| *  *  * [x] [x] [x]|
| *  *  * [x] [x] [x]|    Row-cycling in the nblr-by-nblc [x] blocks.
| *  *  * [x] [x] [x]|    Row-cyclic pivoting inside each [x] block.
|[x] [x] [x] *  *  * |
|[x] [x] [x] *  *  * |
|[x] [x] [x] *  *  * |
```

In terms of the columns of A, the first N1 columns are rotated 'against' the remaining N-N1 columns, trying to increase the angle between the corresponding subspaces. The off-diagonal block is N1-by(N-N1) and it is tiled using quadratic tiles of side KBL. Here, KBL is a tunning parmeter. The number of sweeps is given in NSWEEP and the orthogonality threshold is given in TOL.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgsvj1(JOBV, M, N, N1, A, LDA, D, SVA, MV, V, LDV, EPS, SFMIN, TOL,
                  NSWEEP, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cgsvj1_(const char *jobv, const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *n1, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_singlecomplex_t *d, float *sva,
             const armpl_int_t *mv, armpl_singlecomplex_t *v,
             const armpl_int_t *ldv, const float *eps, const float *sfmin,
             const float *tol, const armpl_int_t *nsweep,
             armpl_singlecomplex_t *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

JOBV Input parameter.

JOBV is CHARACTER*1

Specifies whether the output from this procedure is used to compute the matrix V: = 'V': the product of the Jacobi rotations is accumulated by postmulyiplying the N-by-N array V. (See the description of V.) = 'A': the product of the Jacobi rotations is accumulated by postmulyiplying the MV-by-N array V. (See the descriptions of MV and V.) = 'N': the Jacobi rotations are not accumulated.

M Input parameter.

M is INTEGER

The number of rows of the input matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the input matrix A. M >= N >= 0.

N1 Input parameter.

N1 is INTEGER

N1 specifies the 2 x 2 block partition, the first N1 columns are rotated 'against' the remaining N-N1 columns of A.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, M-by-N matrix A, such that A*diag(D) represents the input matrix. On exit, A_onexit * D_onexit represents the input matrix A*diag(D) post-multiplied by a sequence of Jacobi rotations, where the rotation threshold and the total number of sweeps are given in TOL and NSWEEP, respectively. (See the descriptions of N1, D, TOL and NSWEEP.)

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

D Input and output parameter.

D is COMPLEX

D is an array, dimension (N). The array D accumulates the scaling factors from the fast scaled Jacobi rotations. On entry, A*diag(D) represents the input matrix. On exit, A_onexit*diag(D_onexit) represents the input matrix post-multiplied by a sequence of Jacobi rotations, where the rotation threshold and the total number of sweeps are given in TOL and NSWEEP, respectively. (See the descriptions of N1, A, TOL and NSWEEP.)

SVA Input and output parameter.

SVA is REAL

SVA is an array, dimension (N). On entry, SVA contains the Euclidean norms of the columns of the matrix A*diag(D). On exit, SVA contains the Euclidean norms of the columns of the matrix onexit*diag(D_onexit).

MV Input parameter.

MV is INTEGER

If JOBV .EQ. 'A', then MV rows of V are post-multipled by a sequence of Jacobi rotations. If JOBV = 'N', then MV is not referenced.

V Input and output parameter.

V is COMPLEX

V is an array, dimension (LDV, N). If JOBV .EQ. 'V' then N rows of V are post-multipled by a sequence of Jacobi rotations. If JOBV .EQ. 'A' then MV rows of V are post-multipled by a sequence of Jacobi rotations. If JOBV = 'N', then V is not referenced.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V, LDV >= 1. If JOBV = 'V', LDV .GE. N. If JOBV = 'A', LDV .GE. MV.

EPS Input parameter.

EPS is REAL

EPS = SLAMCH('Epsilon')

SFMIN Input parameter.

SFMIN is REAL

SFMIN = SLAMCH('Safe Minimum')

TOL Input parameter.

TOL is REAL

TOL is the threshold for Jacobi rotations. For a pair A(:,p), A(:,q) of pivot columns, the Jacobi rotation is applied only if ABS(COS(angle(A(:,p),A(:,q)))) .GT. TOL.

NSWEEP Input parameter.

NSWEEP is INTEGER

NSWEEP is the number of sweeps of Jacobi rotations to be performed.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

LWORK is the dimension of WORK. LWORK .GE. M.

INFO Output parameter.

INFO is INTEGER

= 0 : successful exit. < 0 : if INFO = -i, then the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dgsvj1*, *sgsvj1* and *zgsvj1*.

4.17.17 cgtts2

cgtts2 solves one of the systems of equations

```
A * X = B,  A**T * X = B,  or  A**H * X = B,
```

with a tridiagonal matrix A using the LU factorization computed by CGTTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine cgtts2(ITRANS, N, NRHS, DL, D, DU, DU2, IPIV, B, LDB)
```

C specification:

```
#include "armpl.h"

void cgtts2_(const armpl_int_t *itrans, const armpl_int_t *n,
             const armpl_int_t *nrhs, const armpl_singlecomplex_t *dl,
             const armpl_singlecomplex_t *d, const armpl_singlecomplex_t *du,
             const armpl_singlecomplex_t *du2, const armpl_int_t *ipiv,
             armpl_singlecomplex_t *b, const armpl_int_t *ldb);
```

Parameters

ITRANS Input parameter.

ITRANS is INTEGER

Specifies the form of the system of equations. = 0: A * X = B (No transpose) = 1: A^T * X = B (Transpose) = 2: A^H * X = B (Conjugate transpose)

N Input parameter.

N is INTEGER

The order of the matrix A.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

DL Input parameter.

DL is COMPLEX

DL is an array, dimension (N-1). The (n-1) multipliers that define the matrix L from the LU factorization of A.

D Input parameter.

D is COMPLEX

D is an array, dimension (N). The n diagonal elements of the upper triangular matrix U from the LU factorization of A.

DU Input parameter.

DU is COMPLEX

DU is an array, dimension (N-1). The (n-1) elements of the first super-diagonal of U.

DU2 Input parameter.

DU2 is COMPLEX

DU2 is an array, dimension (N-2). The (n-2) elements of the second super-diagonal of U.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices; for 1 <= i <= n, row i of the matrix was interchanged with row IPIV(i). IPIV(i) will always be either i or i+1; IPIV(i) = i indicates a row interchange was not required.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the matrix of right hand side vectors B. On exit, B is overwritten by the solution vectors X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Related Information

For this routine in other precisions, please see *dgtts2*, *sgtts2* and *zgtts2*.

4.17.18 chegs2

chegs2 reduces a complex Hermitian-definite generalized eigenproblem to standard form.

If ITYPE = 1, the problem is A*x = lambda*B*x, and A is overwritten by inv(U^H)*A*inv(U) or inv(L)*A*inv(L^H)

If ITYPE = 2 or 3, the problem is A*B*x = lambda*x or B*A*x = lambda*x, and A is overwritten by U*A*U^H or L^H *A*L.

B must have been previously factorized as U^H *U or L*L^H by ZPOTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine chegs2(ITYPE, UPLO, N, A, LDA, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void chegs2_(const armpl_int_t *itype, const char *uplo, const armpl_int_t *n,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const armpl_singlecomplex_t *b, const armpl_int_t *ldb,
             armpl_int_t *info, ... );
```

Parameters

ITYPE Input parameter.

ITYPE is INTEGER

= 1: compute inv(U^H)*A*inv(U) or inv(L)*A*inv(L^H); = 2 or 3: compute U*A*U^H or L^H *A*L.

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the Hermitian matrix A is stored, and how B has been factorized. = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading n by n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n by n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the transformed matrix, stored in the same format as A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, N). The triangular factor from the Cholesky factorization of B, as returned by CPOTRF.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *zhegs2*.

4.17.19 cheswapr

CHESWAPR applies an elementary permutation on the rows and the columns of a hermitian matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine cheswapr(UPLO, N, A, LDA, I1, I2)
```

C specification:

```
#include "armpl.h"

void cheswapr_(const char *uplo, const armpl_int_t *n,
               armpl_singlecomplex_t *a, const armpl_int_t *lda,
               const armpl_int_t *i1, const armpl_int_t *i2, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the NB diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CSYTRF.

On exit, if INFO = 0, the (symmetric) inverse of the original matrix. If UPLO = 'U', the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; if UPLO = 'L' the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

I1 Input parameter.

I1 is INTEGER

Index of the first row to swap

I2 Input parameter.

I2 is INTEGER

Index of the second row to swap

Related Information

For this routine in other precisions, please see *zheswapr*. It also exists with a native C interface as *LAPACKE_cheswapr*.

4.17.20 chetd2

chetd2 reduces a complex Hermitian matrix A to real symmetric tridiagonal form T by a unitary similarity transformation: $Q^H * A * Q = T$.

Syntax

Fortran specification:

```
use armpl_library

subroutine chetd2(UPLO, N, A, LDA, D, E, TAU, INFO)
```

C specification:

```
#include "armpl.h"

void chetd2_(const char *uplo, const armpl_int_t *n, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, float *d, float *e,
             armpl_singlecomplex_t *tau, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the Hermitian matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, if UPLO = 'U', the diagonal and first superdiagonal of A are overwritten by the corresponding elements of the tridiagonal matrix T, and the elements above the first superdiagonal, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors; if UPLO = 'L', the diagonal and first subdiagonal of A are over- written by the corresponding elements of the tridiagonal matrix T, and the elements below the first subdiagonal, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

D Output parameter.

D is REAL

D is an array, dimension (N). The diagonal elements of the tridiagonal matrix T: D(i) = A(i,i).

E Output parameter.

E is REAL

E is an array, dimension (N-1). The off-diagonal elements of the tridiagonal matrix T: E(i) = A(i,i+1) if UPLO = 'U', E(i) = A(i+1,i) if UPLO = 'L'.

TAU Output parameter.

TAU is COMPLEX

TAU is an array, dimension (N-1). The scalar factors of the elementary reflectors (see Further Details).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *zhetd2*.

4.17.21 chetf2

chetf2 computes the factorization of a complex Hermitian matrix A using the Bunch-Kaufman diagonal pivoting method:

```
A = U*D*U**H  or  A = L*D*L**H
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, U^H is the conjugate transpose of U, and D is Hermitian and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the unblocked version of the algorithm, calling Level 2 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine chetf2(UPLO, N, A, LDA, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void chetf2_(const char *uplo, const armpl_int_t *n, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *ipiv, armpl_int_t *info,
             ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the Hermitian matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L (see below for further details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D.

If UPLO = 'U': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block.

If UPLO = 'L': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value > 0: if INFO = k, D(k,k) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *zhetf2*.

4.17.22 chetf2_rook

CHETF2_ROOK computes the factorization of a complex Hermitian matrix A using the bounded Bunch-Kaufman ("rook") diagonal pivoting method:

```
A = U*D*U**H  or  A = L*D*L**H
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, U^H is the conjugate transpose of U, and D is Hermitian and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the unblocked version of the algorithm, calling Level 2 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine chetf2_rook(UPLO, N, A, LDA, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void chetf2_rook_(const char *uplo, const armpl_int_t *n,
                  armpl_singlecomplex_t *a, const armpl_int_t *lda,
                  armpl_int_t *ipiv, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the Hermitian matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L (see below for further details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D.

If UPLO = 'U': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k-1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k-1 and -IPIV(k-1) were inerchaged, D(k-1:k,k-1:k) is a 2-by-2 diagonal block.

If UPLO = 'L': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k+1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k+1 and -IPIV(k+1) were inerchaged, D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value > 0: if INFO = k, D(k,k) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *zhetf2_rook*.

4.17.23 chfrk

Level 3 BLAS like routine for C in RFP Format.

chfrk performs one of the Hermitian rank–k operations

```
C := alpha*A*A**H + beta*C,
```

or

```
C := alpha*A**H*A + beta*C,
```

where alpha and beta are real scalars, C is an n–by–n Hermitian matrix and A is an n–by–k matrix in the first case and a k–by–n matrix in the second case.

Syntax

Fortran specification:

```
use armpl_library

subroutine chfrk(TRANSR, UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C)
```

C specification:

```
#include "armpl.h"

void chfrk_(const char *transr, const char *uplo, const char *trans,
            const armpl_int_t *n, const armpl_int_t *k, const float *alpha,
            const armpl_singlecomplex_t *a, const armpl_int_t *lda,
            const float *beta, armpl_singlecomplex_t *c, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': The Normal Form of RFP A is stored; = 'C': The Conjugate-transpose Form of RFP A is stored.

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the array C is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of C is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of C is to be referenced.

Unchanged on exit.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' C := alpha*A*A^H + beta*C.

TRANS = 'C' or 'c' C := alpha*A^H * A + beta*C.

Unchanged on exit.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix C. N must be at least zero. Unchanged on exit.

K Input parameter.

K is INTEGER

On entry with TRANS = 'N' or 'n', K specifies the number of columns of the matrix A, and on entry with TRANS = 'C' or 'c', K specifies the number of rows of the matrix A. K must be at least zero. Unchanged on exit.

ALPHA Input parameter.

ALPHA is REAL

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA,ka). where KA is K when TRANS = 'N' or 'n', and is N otherwise. Before entry with TRANS = 'N' or 'n', the leading N–by–K part of the array A must contain the matrix A, otherwise the leading K–by–N part of the array A must contain the matrix A. Unchanged on exit.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDA must be at least max(1, n), otherwise LDA must be at least max(1, k). Unchanged on exit.

BETA Input parameter.

BETA is REAL

On entry, BETA specifies the scalar beta. Unchanged on exit.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (N*(N+1)/2). On entry, the matrix A in RFP Format. RFP Format is described by TRANSR, UPLO and N. Note that the imaginary parts of the diagonal elements need not be set, they are assumed to be zero, and on exit they are set to zero.

Related Information

For this routine in other precisions, please see *zhfrk*. It also exists with a native C interface as *LAPACKE_chfrk*.

4.17.24 cla_gbamv

CLA_GBAMV performs one of the matrix-vector operations

```
      y := alpha*abs(A)*abs(x) + beta*abs(y),
or    y := alpha*abs(A)**T*abs(x) + beta*abs(y),
```

where alpha and beta are scalars, x and y are vectors and A is an m by n matrix.

This function is primarily used in calculating error bounds. To protect against underflow during evaluation, components in the resulting vector are perturbed away from zero by (N+1) times the underflow threshold. To prevent unnecessarily large errors for block-structure embedded in general matrices, "symbolically" zero components are not perturbed. A zero entry is considered "symbolic" if all multiplications involved in computing that entry have at least one zero multiplicand.

Syntax

Fortran specification:

```
use armpl_library

subroutine cla_gbamv(TRANS, M, N, KL, KU, ALPHA, AB, LDAB, X, INCX, BETA, Y,
                     INCY)
```

C specification:

```
#include "armpl.h"

void cla_gbamv_(const armpl_int_t *trans, const armpl_int_t *m,
                const armpl_int_t *n, const armpl_int_t *kl,
                const armpl_int_t *ku, const float *alpha,
                const armpl_singlecomplex_t *ab, const armpl_int_t *ldab,
                const armpl_singlecomplex_t *x, const armpl_int_t *incx,
                const float *beta, float *y, const armpl_int_t *incy);
```

Parameters

TRANS Input parameter.

TRANS is INTEGER

On entry, TRANS specifies the operation to be performed as follows:

BLAS_NO_TRANS y := alpha*abs(A)*abs(x) + beta*abs(y) BLAS_TRANS y := alpha*abs(A^T)*abs(x) + beta*abs(y) BLAS_CONJ_TRANS y := alpha*abs(A^T)*abs(x) + beta*abs(y)

Unchanged on exit.

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of the matrix A. M must be at least zero. Unchanged on exit.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix A. N must be at least zero. Unchanged on exit.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

ALPHA Input parameter.

ALPHA is REAL

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

AB Input parameter.

AB is COMPLEX

AB is an array, dimension (LDAB,n). Before entry, the leading m by n part of the array AB must contain the matrix of coefficients. Unchanged on exit.

LDAB Input parameter.

LDAB is INTEGER

On entry, LDAB specifies the first dimension of AB as declared in the calling (sub) program. LDAB must be at least max(1, m). Unchanged on exit.

X Input parameter.

X is COMPLEX

X is an array, dimension. (1 + (n - 1)*abs(INCX)) when TRANS = 'N' or 'n' and at least (1 + (m - 1)*abs(INCX)) otherwise. Before entry, the incremented array X must contain the vector x. Unchanged on exit.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

BETA Input parameter.

BETA is REAL

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input. Unchanged on exit.

Y Input and output parameter.

Y is REAL

Y is an array, dimension. (1 + (m - 1)*abs(INCY)) when TRANS = 'N' or 'n' and at least (1 + (n - 1)*abs(INCY)) otherwise. Before entry with BETA non-zero, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero. Unchanged on exit.

Level 2 Blas routine.

Related Information

For this routine in other precisions, please see *dla_gbamv*, *sla_gbamv* and *zla_gbamv*.

4.17.25 cla_gbrcond_c

```
CLA_GBRCOND_C Computes the infinity norm condition number of
   op(A) * inv(diag(C)) where C is a REAL vector.
```

Syntax

Fortran specification:

```
use armpl_library

real function cla_gbrcond_c(TRANS, N, KL, KU, AB, LDAB, AFB, LDAFB, IPIV, C,
                            CAPPLY, INFO, WORK, RWORK)
```

C specification:

```
#include "armpl.h"

float cla_gbrcond_c_(const char *trans, const armpl_int_t *n,
                     const armpl_int_t *kl, const armpl_int_t *ku,
                     const armpl_singlecomplex_t *ab, const armpl_int_t *ldab,
                     const armpl_singlecomplex_t *afb,
                     const armpl_int_t *ldafb, const armpl_int_t *ipiv,
                     const float *c, const armpl_int_t *capply,
                     armpl_int_t *info, const armpl_singlecomplex_t *work,
                     const float *rwork, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate Transpose = Transpose)

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

AB Input parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(KU+1+i-j,j) = A(i,j) for max(1,j-KU)<=i<=min(N,j+kl)

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KL+KU+1.

AFB Input parameter.

AFB is COMPLEX

AFB is an array, dimension (LDAFB, N). Details of the LU factorization of the band matrix A, as computed by CGBTRF. U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1.

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AFB. LDAFB >= 2*KL+KU+1.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from the factorization A = P*L*U as computed by CGBTRF; row i of the matrix was interchanged with row IPIV(i).

C Input parameter.

C is REAL

C is an array, dimension (N). The vector C in the formula op(A) * inv(diag(C)).

CAPPLY Input parameter.

CAPPLY is LOGICAL

If .TRUE. then access the vector C in the formula above.

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. i > 0: The ith argument is invalid.

WORK Input parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N).. Workspace.

RWORK Input parameter.

RWORK is REAL

RWORK is an array, dimension (N).. Workspace.

Related Information

For this routine in other precisions, please see *zla_gbrcond_c*.

4.17.26 cla_gbrcond_x

```
CLA_GBRCOND_X Computes the infinity norm condition number of
   op(A) * diag(X) where X is a COMPLEX vector.
```

Syntax

Fortran specification:

```
use armpl_library

real function cla_gbrcond_x(TRANS, N, KL, KU, AB, LDAB, AFB, LDAFB, IPIV, X,
                            INFO, WORK, RWORK)
```

C specification:

```
#include "armpl.h"

float cla_gbrcond_x_(const char *trans, const armpl_int_t *n,
                     const armpl_int_t *kl, const armpl_int_t *ku,
                     const armpl_singlecomplex_t *ab, const armpl_int_t *ldab,
                     const armpl_singlecomplex_t *afb,
                     const armpl_int_t *ldafb, const armpl_int_t *ipiv,
                     const armpl_singlecomplex_t *x, armpl_int_t *info,
                     const armpl_singlecomplex_t *work, const float *rwork,
                     ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate Transpose = Transpose)

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

AB Input parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(KU+1+i-j,j) = A(i,j) for max(1,j-KU)<=i<=min(N,j+kl)

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KL+KU+1.

AFB Input parameter.

AFB is COMPLEX

AFB is an array, dimension (LDAFB, N). Details of the LU factorization of the band matrix A, as computed by CGBTRF. U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1.

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AFB. LDAFB >= 2*KL+KU+1.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from the factorization A = P*L*U as computed by CGBTRF; row i of the matrix was interchanged with row IPIV(i).

X Input parameter.

X is COMPLEX

X is an array, dimension (N). The vector X in the formula op(A) * diag(X).

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. i > 0: The ith argument is invalid.

WORK Input parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N).. Workspace.

RWORK Input parameter.

RWORK is REAL

RWORK is an array, dimension (N).. Workspace.

Related Information

For this routine in other precisions, please see *zla_gbrcond_x*.

4.17.27 cla_gbrfsx_extended

cla_gbrfsx_extended improves the computed solution to a system of linear equations by performing extra-precise iterative refinement and provides error bounds and backward error estimates for the solution. This subroutine is called by CGBRFSX to perform iterative refinement. In addition to normwise error bound, the code provides maximum componentwise error bound if possible. See comments for ERR_BNDS_NORM and ERR_BNDS_COMP for details of the error bounds. Note that this subroutine is only resonsible for setting the second fields of ERR_BNDS_NORM and ERR_BNDS_COMP.

Syntax

Fortran specification:

```
use armpl_library

subroutine cla_gbrfsx_extended(PREC_TYPE, TRANS_TYPE, N, KL, KU, NRHS, AB,
                               LDAB, AFB, LDAFB, IPIV, COLEQU, C, B, LDB, Y,
                               LDY, BERR_OUT, N_NORMS, ERR_BNDS_NORM,
                               ERR_BNDS_COMP, RES, AYB, DY, Y_TAIL, RCOND,
                               ITHRESH, RTHRESH, DZ_UB, IGNORE_CWISE, INFO)
```

C specification:

```
#include "armpl.h"

void cla_gbrfsx_extended_(const armpl_int_t *prec_type,
                          const armpl_int_t *trans_type, const armpl_int_t *n,
                          const armpl_int_t *kl, const armpl_int_t *ku,
                          const armpl_int_t *nrhs,
                          const armpl_singlecomplex_t *ab,
                          const armpl_int_t *ldab,
                          const armpl_singlecomplex_t *afb,
                          const armpl_int_t *ldafb, const armpl_int_t *ipiv,
                          const armpl_int_t *colequ, const float *c,
                          const armpl_singlecomplex_t *b,
                          const armpl_int_t *ldb, armpl_singlecomplex_t *y,
                          const armpl_int_t *ldy, float *berr_out,
                          const armpl_int_t *n_norms, float *err_bnds_norm,
                          float *err_bnds_comp, armpl_singlecomplex_t *res,
                          float *ayb, armpl_singlecomplex_t *dy,
                          armpl_singlecomplex_t *y_tail, const float *rcond,
                          const armpl_int_t *ithresh, const float *rthresh,
                          const float *dz_ub, const armpl_int_t *ignore_cwise,
                          armpl_int_t *info);
```

Parameters

PREC_TYPE Input parameter.

PREC_TYPE is INTEGER

Specifies the intermediate precision to be used in refinement. The value is defined by ILAPREC(P) where P is a CHARACTER and P = 'S': Single = 'D': Double = 'I': Indigenous = 'X', 'E': Extra

TRANS_TYPE Input parameter.

TRANS_TYPE is INTEGER

Specifies the transposition operation on A. The value is defined by ILATRANS(T) where T is a CHARACTER and T = 'N': No transpose = 'T': Transpose = 'C': Conjugate transpose

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0

NRHS Input parameter.

NRHS is INTEGER

The number of right-hand-sides, i.e., the number of columns of the matrix B.

AB Input parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). On entry, the N-by-N matrix AB.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= max(1, N).

AFB Input parameter.

AFB is COMPLEX

AFB is an array, dimension (LDAF, N). The factors L and U from the factorization A = P*L*U as computed by CGBTRF.

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from the factorization A = P*L*U as computed by CGBTRF; row i of the matrix was interchanged with row IPIV(i).

COLEQU Input parameter.

COLEQU is LOGICAL

If .TRUE. then column equilibration was done to A before calling this routine. This is needed to compute the solution and error bounds correctly.

C Input parameter.

C is REAL

C is an array, dimension (N). The column scale factors for A. If COLEQU = .FALSE., C is not accessed. If C is input, each element of C should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). The right-hand-side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Y Input and output parameter.

Y is COMPLEX

Y is an array, dimension (LDY, NRHS). On entry, the solution matrix X, as computed by CGBTRS. On exit, the improved solution matrix Y.

LDY Input parameter.

LDY is INTEGER

The leading dimension of the array Y. LDY >= max(1, N).

BERR_OUT Output parameter.

BERR_OUT is REAL

BERR_OUT is an array, dimension (NRHS). On exit, BERR_OUT(j) contains the componentwise relative backward error for right-hand-side j from the formula max(i) (abs(RES(i)) / (abs(op(A_s))*abs(Y) + abs(B_s)

)(i)) where abs(Z) is the componentwise absolute value of the matrix or vector Z. This is computed by CLA_LIN_BERR.

N_NORMS Input parameter.

N_NORMS is INTEGER

Determines which error bounds to return (see ERR_BNDS_NORM and ERR_BNDS_COMP). If N_NORMS >= 1 return normwise error bounds. If N_NORMS >= 2 return componentwise error bounds.

ERR_BNDS_NORM Input and output parameter.

ERR_BNDS_NORM is REAL

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

This subroutine is only responsible for setting the second field above. See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Input and output parameter.

ERR_BNDS_COMP is REAL

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:,N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

This subroutine is only responsible for setting the second field above. See Lapack Working Note 165 for further details and extra cautions.

RES Input parameter.

RES is COMPLEX

RES is an array, dimension (N). Workspace to hold the intermediate residual.

AYB Input parameter.

AYB is REAL

AYB is an array, dimension (N). Workspace.

DY Input parameter.

DY is COMPLEX

DY is an array, dimension (N). Workspace to hold the intermediate solution.

Y_TAIL Input parameter.

Y_TAIL is COMPLEX

Y_TAIL is an array, dimension (N). Workspace to hold the trailing bits of the intermediate solution.

RCOND Input parameter.

RCOND is REAL

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

ITHRESH Input parameter.

ITHRESH is INTEGER

The maximum number of residual computations allowed for refinement. The default is 10. For 'aggressive' set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in ERR_BNDS_NORM and ERR_BNDS_COMP may no longer be trustworthy.

RTHRESH Input parameter.

RTHRESH is REAL

Determines when to stop refinement if the error estimate stops decreasing. Refinement will stop when the next solution no longer satisfies norm(dx_{i+1}) < RTHRESH * norm(dx_i) where norm(Z) is the infinity norm of Z. RTHRESH satisfies 0 < RTHRESH <= 1. The default value is 0.5. For 'aggressive' set to 0.9 to permit convergence on extremely ill-conditioned matrices. See LAWN 165 for more details.

DZ_UB Input parameter.

DZ_UB is REAL

Determines when to start considering componentwise convergence. Componentwise convergence is only considered after each component of the solution Y is stable, which we definte as the relative change in each component being less than DZ_UB. The default value is 0.25, requiring the first bit to be stable. See LAWN 165 for more details.

IGNORE_CWISE Input parameter.

IGNORE_CWISE is LOGICAL

If .TRUE. then ignore componentwise convergence. Default value is .FALSE..

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. < 0: if INFO = -i, the ith argument to CGBTRS had an illegal value

Related Information

For this routine in other precisions, please see *dla_gbrfsx_extended*, *sla_gbrfsx_extended* and *zla_gbrfsx_extended*.

4.17.28 cla_gbrpvgrw

CLA_GBRPVGRW computes the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If this is much less than 1, the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, estimated condition numbers, and error bounds could be unreliable.

Syntax

Fortran specification:

```
use armpl_library

real function cla_gbrpvgrw(N, KL, KU, NCOLS, AB, LDAB, AFB, LDAFB)
```

C specification:

```
#include "armpl.h"

float cla_gbrpvgrw_(const armpl_int_t *n, const armpl_int_t *kl,
                    const armpl_int_t *ku, const armpl_int_t *ncols,
                    const armpl_singlecomplex_t *ab, const armpl_int_t *ldab,
                    const armpl_singlecomplex_t *afb,
                    const armpl_int_t *ldafb);
```

Parameters

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

NCOLS Input parameter.

NCOLS is INTEGER

The number of columns of the matrix A. NCOLS >= 0.

AB Input parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(KU+1+i-j,j) = A(i,j) for max(1,j-KU)<=i<=min(N,j+kl)

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KL+KU+1.

AFB Input parameter.

AFB is COMPLEX

AFB is an array, dimension (LDAFB, N). Details of the LU factorization of the band matrix A, as computed by CGBTRF. U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1.

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AFB. LDAFB >= 2*KL+KU+1.

Related Information

For this routine in other precisions, please see *dla_gbrpvgrw*, *sla_gbrpvgrw* and *zla_gbrpvgrw*.

4.17.29 cla_geamv

CLA_GEAMV performs one of the matrix-vector operations

```
      y := alpha*abs(A)*abs(x) + beta*abs(y),
or    y := alpha*abs(A)**T*abs(x) + beta*abs(y),
```

where alpha and beta are scalars, x and y are vectors and A is an m by n matrix.

This function is primarily used in calculating error bounds. To protect against underflow during evaluation, components in the resulting vector are perturbed away from zero by (N+1) times the underflow threshold. To prevent unnecessarily large errors for block-structure embedded in general matrices, "symbolically" zero components are not perturbed. A zero entry is considered "symbolic" if all multiplications involved in computing that entry have at least one zero multiplicand.

Syntax

Fortran specification:

```
use armpl_library

subroutine cla_geamv(TRANS, M, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
```

C specification:

```
#include "armpl.h"

void cla_geamv_(const armpl_int_t *trans, const armpl_int_t *m,
                const armpl_int_t *n, const float *alpha,
                const armpl_singlecomplex_t *a, const armpl_int_t *lda,
                const armpl_singlecomplex_t *x, const armpl_int_t *incx,
                const float *beta, float *y, const armpl_int_t *incy);
```

Parameters

TRANS Input parameter.

TRANS is INTEGER

On entry, TRANS specifies the operation to be performed as follows:

BLAS_NO_TRANS y := alpha*abs(A)*abs(x) + beta*abs(y) BLAS_TRANS y := alpha*abs(A^T)*abs(x) + beta*abs(y) BLAS_CONJ_TRANS y := alpha*abs(A^T)*abs(x) + beta*abs(y)

Unchanged on exit.

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of the matrix A. M must be at least zero. Unchanged on exit.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix A. N must be at least zero. Unchanged on exit.

ALPHA Input parameter.

ALPHA is REAL

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA,n). Before entry, the leading m by n part of the array A must contain the matrix of coefficients. Unchanged on exit.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, m). Unchanged on exit.

X Input parameter.

X is COMPLEX

X is an array, dimension. (1 + (n - 1)*abs(INCX)) when TRANS = 'N' or 'n' and at least (1 + (m - 1)*abs(INCX)) otherwise. Before entry, the incremented array X must contain the vector x. Unchanged on exit.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

BETA Input parameter.

BETA is REAL

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input. Unchanged on exit.

Y Input and output parameter.

Y is REAL

Y is an array, dimension. (1 + (m - 1)*abs(INCY)) when TRANS = 'N' or 'n' and at least (1 + (n - 1)*abs(INCY)) otherwise. Before entry with BETA non-zero, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero. Unchanged on exit.

Level 2 Blas routine.

Related Information

For this routine in other precisions, please see *dla_geamv*, *sla_geamv* and *zla_geamv*.

4.17.30 cla_gercond_c

```
CLA_GERCOND_C computes the infinity norm condition number of
   op(A) * inv(diag(C)) where C is a REAL vector.
```

Syntax

Fortran specification:

```
use armpl_library

real function cla_gercond_c(TRANS, N, A, LDA, AF, LDAF, IPIV, C, CAPPLY, INFO,
                            WORK, RWORK)
```

C specification:

```
#include "armpl.h"

float cla_gercond_c_(const char *trans, const armpl_int_t *n,
                     const armpl_singlecomplex_t *a, const armpl_int_t *lda,
                     const armpl_singlecomplex_t *af, const armpl_int_t *ldaf,
                     const armpl_int_t *ipiv, const float *c,
                     const armpl_int_t *capply, armpl_int_t *info,
                     const armpl_singlecomplex_t *work, const float *rwork,
                     ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate Transpose = Transpose)

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX

AF is an array, dimension (LDAF, N). The factors L and U from the factorization A = P*L*U as computed by CGETRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from the factorization A = P*L*U as computed by CGETRF; row i of the matrix was interchanged with row IPIV(i).

C Input parameter.

C is REAL

C is an array, dimension (N). The vector C in the formula op(A) * inv(diag(C)).

CAPPLY Input parameter.

CAPPLY is LOGICAL

If .TRUE. then access the vector C in the formula above.

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. i > 0: The ith argument is invalid.

WORK Input parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N).. Workspace.

RWORK Input parameter.

RWORK is REAL

RWORK is an array, dimension (N).. Workspace.

Related Information

For this routine in other precisions, please see *zla_gercond_c*.

4.17.31 cla_gercond_x

```
CLA_GERCOND_X computes the infinity norm condition number of
   op(A) * diag(X) where X is a COMPLEX vector.
```

Syntax

Fortran specification:

```
use armpl_library

real function cla_gercond_x(TRANS, N, A, LDA, AF, LDAF, IPIV, X, INFO, WORK,
                            RWORK)
```

C specification:

```
#include "armpl.h"

float cla_gercond_x_(const char *trans, const armpl_int_t *n,
                     const armpl_singlecomplex_t *a, const armpl_int_t *lda,
                     const armpl_singlecomplex_t *af, const armpl_int_t *ldaf,
                     const armpl_int_t *ipiv, const armpl_singlecomplex_t *x,
                     armpl_int_t *info, const armpl_singlecomplex_t *work,
                     const float *rwork, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate Transpose = Transpose)

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX

AF is an array, dimension (LDAF, N). The factors L and U from the factorization A = P*L*U as computed by CGETRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from the factorization A = P*L*U as computed by CGETRF; row i of the matrix was interchanged with row IPIV(i).

X Input parameter.

X is COMPLEX

X is an array, dimension (N). The vector X in the formula op(A) * diag(X).

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. i > 0: The ith argument is invalid.

WORK Input parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N).. Workspace.

RWORK Input parameter.

RWORK is REAL

RWORK is an array, dimension (N).. Workspace.

Related Information

For this routine in other precisions, please see *zla_gercond_x*.

4.17.32 cla_gerfsx_extended

cla_gerfsx_extended improves the computed solution to a system of linear equations by performing extra-precise iterative refinement and provides error bounds and backward error estimates for the solution. This subroutine is called by CGERFSX to perform iterative refinement. In addition to normwise error bound, the code provides maximum componentwise error bound if possible. See comments for ERRS_N and ERRS_C for details of the error bounds. Note that this subroutine is only resonsible for setting the second fields of ERRS_N and ERRS_C.

Syntax

Fortran specification:

```
use armpl_library

subroutine cla_gerfsx_extended(PREC_TYPE, TRANS_TYPE, N, NRHS, A, LDA, AF,
                               LDAF, IPIV, COLEQU, C, B, LDB, Y, LDY,
                               BERR_OUT, N_NORMS, ERRS_N, ERRS_C, RES, AYB,
                               DY, Y_TAIL, RCOND, ITHRESH, RTHRESH, DZ_UB,
                               IGNORE_CWISE, INFO)
```

C specification:

```
#include "armpl.h"

void cla_gerfsx_extended_(const armpl_int_t *prec_type,
                          const armpl_int_t *trans_type, const armpl_int_t *n,
                          const armpl_int_t *nrhs,
                          const armpl_singlecomplex_t *a,
                          const armpl_int_t *lda,
```

(continues on next page)

(continued from previous page)

```
                        const armpl_singlecomplex_t *af,
                        const armpl_int_t *ldaf, const armpl_int_t *ipiv,
                        const armpl_int_t *colequ, const float *c,
                        const armpl_singlecomplex_t *b,
                        const armpl_int_t *ldb, armpl_singlecomplex_t *y,
                        const armpl_int_t *ldy, float *berr_out,
                        const armpl_int_t *n_norms, float *errs_n,
                        float *errs_c, armpl_singlecomplex_t *res,
                        float *ayb, armpl_singlecomplex_t *dy,
                        armpl_singlecomplex_t *y_tail, const float *rcond,
                        const armpl_int_t *ithresh, const float *rthresh,
                        const float *dz_ub, const armpl_int_t *ignore_cwise,
                        armpl_int_t *info);
```

Parameters

PREC_TYPE Input parameter.

PREC_TYPE is INTEGER

Specifies the intermediate precision to be used in refinement. The value is defined by ILAPREC(P) where P is a CHARACTER and P = 'S': Single = 'D': Double = 'I': Indigenous = 'X', 'E': Extra

TRANS_TYPE Input parameter.

TRANS_TYPE is INTEGER

Specifies the transposition operation on A. The value is defined by ILATRANS(T) where T is a CHARACTER and T = 'N': No transpose = 'T': Transpose = 'C': Conjugate transpose

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right-hand-sides, i.e., the number of columns of the matrix B.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX

AF is an array, dimension (LDAF, N). The factors L and U from the factorization A = P*L*U as computed by CGETRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from the factorization A = P*L*U as computed by CGETRF; row i of the matrix was interchanged with row IPIV(i).

COLEQU Input parameter.

COLEQU is LOGICAL

If .TRUE. then column equilibration was done to A before calling this routine. This is needed to compute the solution and error bounds correctly.

C Input parameter.

C is REAL

C is an array, dimension (N). The column scale factors for A. If COLEQU = .FALSE., C is not accessed. If C is input, each element of C should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). The right-hand-side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Y Input and output parameter.

Y is COMPLEX

Y is an array, dimension (LDY, NRHS). On entry, the solution matrix X, as computed by CGETRS. On exit, the improved solution matrix Y.

LDY Input parameter.

LDY is INTEGER

The leading dimension of the array Y. LDY >= max(1, N).

BERR_OUT Output parameter.

BERR_OUT is REAL

BERR_OUT is an array, dimension (NRHS). On exit, BERR_OUT(j) contains the componentwise relative backward error for right-hand-side j from the formula max(i) (abs(RES(i)) / (abs(op(A_s))*abs(Y) + abs(B_s))(i)) where abs(Z) is the componentwise absolute value of the matrix or vector Z. This is computed by CLA_LIN_BERR.

N_NORMS Input parameter.

N_NORMS is INTEGER

Determines which error bounds to return (see ERRS_N and ERRS_C). If N_NORMS >= 1 return normwise error bounds. If N_NORMS >= 2 return componentwise error bounds.

ERRS_N Input and output parameter.

ERRS_N is REAL

ERRS_N is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERRS_N(i,:) corresponds to the ith right-hand side.

The second index in ERRS_N(:,err) contains the following three fields: err = 1 “Trust/don’t trust” boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch(‘Epsilon’).

err = 2 “Guaranteed” error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch(‘Epsilon’). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch(‘Epsilon’) to determine if the error estimate is “guaranteed”. These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

This subroutine is only responsible for setting the second field above. See Lapack Working Note 165 for further details and extra cautions.

ERRS_C Input and output parameter.

ERRS_C is REAL

ERRS_C is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERRS_C is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:,N_ERR_BNDS) entries are returned.

The first index in ERRS_C(i,:) corresponds to the ith right-hand side.

The second index in ERRS_C(:,err) contains the following three fields: err = 1 “Trust/don’t trust” boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch(‘Epsilon’).

err = 2 “Guaranteed” error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch(‘Epsilon’). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch(‘Epsilon’) to determine if the error estimate is “guaranteed”. These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

This subroutine is only responsible for setting the second field above. See Lapack Working Note 165 for further details and extra cautions.

RES Input parameter.

RES is COMPLEX

RES is an array, dimension (N). Workspace to hold the intermediate residual.

AYB Input parameter.

AYB is REAL

AYB is an array, dimension (N). Workspace.

DY Input parameter.

DY is COMPLEX

DY is an array, dimension (N). Workspace to hold the intermediate solution.

Y_TAIL Input parameter.

Y_TAIL is COMPLEX

Y_TAIL is an array, dimension (N). Workspace to hold the trailing bits of the intermediate solution.

RCOND Input parameter.

RCOND is REAL

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

ITHRESH Input parameter.

ITHRESH is INTEGER

The maximum number of residual computations allowed for refinement. The default is 10. For 'aggressive' set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in ERRS_N and ERRS_C may no longer be trustworthy.

RTHRESH Input parameter.

RTHRESH is REAL

Determines when to stop refinement if the error estimate stops decreasing. Refinement will stop when the next solution no longer satisfies norm(dx_{i+1}) < RTHRESH * norm(dx_i) where norm(Z) is the infinity norm of Z. RTHRESH satisfies 0 < RTHRESH <= 1. The default value is 0.5. For 'aggressive' set to 0.9 to permit convergence on extremely ill-conditioned matrices. See LAWN 165 for more details.

DZ_UB Input parameter.

DZ_UB is REAL

Determines when to start considering componentwise convergence. Componentwise convergence is only considered after each component of the solution Y is stable, which we definte as the relative change in each component being less than DZ_UB. The default value is 0.25, requiring the first bit to be stable. See LAWN 165 for more details.

IGNORE_CWISE Input parameter.

IGNORE_CWISE is LOGICAL

If .TRUE. then ignore componentwise convergence. Default value is .FALSE..

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. < 0: if INFO = -i, the ith argument to CGETRS had an illegal value

Related Information

For this routine in other precisions, please see *dla_gerfsx_extended*, *sla_gerfsx_extended* and *zla_gerfsx_extended*.

4.17.33 cla_gerpvgrw

CLA_GERPVGRW computes the reciprocal pivot growth factor norm(A)/norm(U). The “max absolute element” norm is used. If this is much less than 1, the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, estimated condition numbers, and error bounds could be unreliable.

Syntax

Fortran specification:

```
use armpl_library

real function cla_gerpvgrw(N, NCOLS, A, LDA, AF, LDAF)
```

C specification:

```
#include "armpl.h"

float cla_gerpvgrw_(const armpl_int_t *n, const armpl_int_t *ncols,
                    const armpl_singlecomplex_t *a, const armpl_int_t *lda,
                    const armpl_singlecomplex_t *af,
                    const armpl_int_t *ldaf);
```

Parameters

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NCOLS Input parameter.

NCOLS is INTEGER

The number of columns of the matrix A. NCOLS >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX

AF is an array, dimension (LDAF, N). The factors L and U from the factorization A = P*L*U as computed by CGETRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

Related Information

For this routine in other precisions, please see *dla_gerpvgrw*, *sla_gerpvgrw* and *zla_gerpvgrw*.

4.17.34 cla_heamv

CLA_SYAMV performs the matrix-vector operation

```
y := alpha*abs(A)*abs(x) + beta*abs(y),
```

where alpha and beta are scalars, x and y are vectors and A is an n by n symmetric matrix.

This function is primarily used in calculating error bounds. To protect against underflow during evaluation, components in the resulting vector are perturbed away from zero by (N+1) times the underflow threshold. To prevent unnecessarily large errors for block-structure embedded in general matrices, "symbolically" zero components are not perturbed. A zero entry is considered "symbolic" if all multiplications involved in computing that entry have at least one zero multiplicand.

Syntax

Fortran specification:

```
use armpl_library

subroutine cla_heamv(UPLO, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
```

C specification:

```
#include "armpl.h"

void cla_heamv_(const armpl_int_t *uplo, const armpl_int_t *n,
                const float *alpha, const armpl_singlecomplex_t *a,
                const armpl_int_t *lda, const armpl_singlecomplex_t *x,
                const armpl_int_t *incx, const float *beta, float *y,
                const armpl_int_t *incy);
```

Parameters

UPLO Input parameter.

UPLO is INTEGER

On entry, UPLO specifies whether the upper or lower triangular part of the array A is to be referenced as follows:

UPLO = BLAS_UPPER Only the upper triangular part of A is to be referenced.

UPLO = BLAS_LOWER Only the lower triangular part of A is to be referenced.

Unchanged on exit.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix A. N must be at least zero. Unchanged on exit.

ALPHA Input parameter.

ALPHA is REAL .

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, n).. Before entry, the leading m by n part of the array A must contain the matrix of coefficients. Unchanged on exit.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, n). Unchanged on exit.

X Input parameter.

X is COMPLEX

X is an array, dimension. (1 + (n - 1)*abs(INCX)) Before entry, the incremented array X must contain the vector x. Unchanged on exit.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

BETA Input parameter.

BETA is REAL .

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input. Unchanged on exit.

Y Input and output parameter.

Y is REAL

Y is an array, dimension. (1 + (n - 1)*abs(INCY)) Before entry with BETA non-zero, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero. Unchanged on exit.

Related Information

For this routine in other precisions, please see *zla_heamv*.

4.17.35 cla_hercond_c

```
CLA_HERCOND_C computes the infinity norm condition number of
   op(A) * inv(diag(C)) where C is a REAL vector.
```

Syntax

Fortran specification:

```
use armpl_library

real function cla_hercond_c(UPLO, N, A, LDA, AF, LDAF, IPIV, C, CAPPLY, INFO,
                            WORK, RWORK)
```

C specification:

```
#include "armpl.h"

float cla_hercond_c_(const char *uplo, const armpl_int_t *n,
                     const armpl_singlecomplex_t *a, const armpl_int_t *lda,
```

(continues on next page)

(continued from previous page)

```
                    const armpl_singlecomplex_t *af, const armpl_int_t *ldaf,
                    const armpl_int_t *ipiv, const float *c,
                    const armpl_int_t *capply, armpl_int_t *info,
                    const armpl_singlecomplex_t *work, const float *rwork,
                    ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX

AF is an array, dimension (LDAF, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CHETRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CHETRF.

C Input parameter.

C is REAL

C is an array, dimension (N). The vector C in the formula op(A) * inv(diag(C)).

CAPPLY Input parameter.

CAPPLY is LOGICAL

If .TRUE. then access the vector C in the formula above.

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. i > 0: The ith argument is invalid.

WORK Input parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N).. Workspace.

RWORK Input parameter.

RWORK is REAL

RWORK is an array, dimension (N).. Workspace.

Related Information

For this routine in other precisions, please see *zla_hercond_c*.

4.17.36 cla_hercond_x

```
CLA_HERCOND_X computes the infinity norm condition number of
   op(A) * diag(X) where X is a COMPLEX vector.
```

Syntax

Fortran specification:

```
use armpl_library

real function cla_hercond_x(UPLO, N, A, LDA, AF, LDAF, IPIV, X, INFO, WORK,
                            RWORK)
```

C specification:

```
#include "armpl.h"

float cla_hercond_x_(const char *uplo, const armpl_int_t *n,
                     const armpl_singlecomplex_t *a, const armpl_int_t *lda,
                     const armpl_singlecomplex_t *af, const armpl_int_t *ldaf,
                     const armpl_int_t *ipiv, const armpl_singlecomplex_t *x,
                     armpl_int_t *info, const armpl_singlecomplex_t *work,
                     const float *rwork, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX

AF is an array, dimension (LDAF, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CHETRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CHETRF.

X Input parameter.

X is COMPLEX

X is an array, dimension (N). The vector X in the formula op(A) * diag(X).

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. i > 0: The ith argument is invalid.

WORK Input parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N).. Workspace.

RWORK Input parameter.

RWORK is REAL

RWORK is an array, dimension (N).. Workspace.

Related Information

For this routine in other precisions, please see *zla_hercond_x*.

4.17.37 cla_herfsx_extended

cla_herfsx_extended improves the computed solution to a system of linear equations by performing extra-precise iterative refinement and provides error bounds and backward error estimates for the solution. This subroutine is called by CHERFSX to perform iterative refinement. In addition to normwise error bound, the code provides maximum componentwise error bound if possible. See comments for ERR_BNDS_NORM and ERR_BNDS_COMP for details of the error bounds. Note that this subroutine is only resonsible for setting the second fields of ERR_BNDS_NORM and ERR_BNDS_COMP.

Syntax

Fortran specification:

```
use armpl_library

subroutine cla_herfsx_extended(PREC_TYPE, UPLO, N, NRHS, A, LDA, AF, LDAF,
                               IPIV, COLEQU, C, B, LDB, Y, LDY, BERR_OUT,
                               N_NORMS, ERR_BNDS_NORM, ERR_BNDS_COMP, RES,
                               AYB, DY, Y_TAIL, RCOND, ITHRESH, RTHRESH,
                               DZ_UB, IGNORE_CWISE, INFO)
```

C specification:

```
#include "armpl.h"

void cla_herfsx_extended_(const armpl_int_t *prec_type, const char *uplo,
                          const armpl_int_t *n, const armpl_int_t *nrhs,
                          const armpl_singlecomplex_t *a,
                          const armpl_int_t *lda,
                          const armpl_singlecomplex_t *af,
                          const armpl_int_t *ldaf, const armpl_int_t *ipiv,
                          const armpl_int_t *colequ, const float *c,
                          const armpl_singlecomplex_t *b,
                          const armpl_int_t *ldb, armpl_singlecomplex_t *y,
                          const armpl_int_t *ldy, float *berr_out,
                          const armpl_int_t *n_norms, float *err_bnds_norm,
                          float *err_bnds_comp, armpl_singlecomplex_t *res,
                          float *ayb, armpl_singlecomplex_t *dy,
                          armpl_singlecomplex_t *y_tail, const float *rcond,
                          const armpl_int_t *ithresh, const float *rthresh,
                          const float *dz_ub, const armpl_int_t *ignore_cwise,
                          armpl_int_t *info, ... );
```

Parameters

PREC_TYPE Input parameter.

PREC_TYPE is INTEGER

Specifies the intermediate precision to be used in refinement. The value is defined by ILAPREC(P) where P is a CHARACTER and P = 'S': Single = 'D': Double = 'I': Indigenous = 'X', 'E': Extra

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right-hand-sides, i.e., the number of columns of the matrix B.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX

AF is an array, dimension (LDAF, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CHETRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CHETRF.

COLEQU Input parameter.

COLEQU is LOGICAL

If .TRUE. then column equilibration was done to A before calling this routine. This is needed to compute the solution and error bounds correctly.

C Input parameter.

C is REAL

C is an array, dimension (N). The column scale factors for A. If COLEQU = .FALSE., C is not accessed. If C is input, each element of C should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). The right-hand-side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Y Input and output parameter.

Y is COMPLEX

Y is an array, dimension (LDY, NRHS). On entry, the solution matrix X, as computed by CHETRS. On exit, the improved solution matrix Y.

LDY Input parameter.

LDY is INTEGER

The leading dimension of the array Y. LDY >= max(1, N).

BERR_OUT Output parameter.

BERR_OUT is REAL

BERR_OUT is an array, dimension (NRHS). On exit, BERR_OUT(j) contains the componentwise relative backward error for right-hand-side j from the formula max(i) (abs(RES(i)) / (abs(op(A_s))*abs(Y) + abs(B_s))(i)) where abs(Z) is the componentwise absolute value of the matrix or vector Z. This is computed by CLA_LIN_BERR.

N_NORMS Input parameter.

N_NORMS is INTEGER

Determines which error bounds to return (see ERR_BNDS_NORM and ERR_BNDS_COMP). If N_NORMS >= 1 return normwise error bounds. If N_NORMS >= 2 return componentwise error bounds.

ERR_BNDS_NORM Input and output parameter.

ERR_BNDS_NORM is REAL

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

This subroutine is only responsible for setting the second field above. See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Input and output parameter.

ERR_BNDS_COMP is REAL

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:,N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix

Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

This subroutine is only responsible for setting the second field above. See Lapack Working Note 165 for further details and extra cautions.

RES Input parameter.

RES is COMPLEX

RES is an array, dimension (N). Workspace to hold the intermediate residual.

AYB Input parameter.

AYB is REAL

AYB is an array, dimension (N). Workspace.

DY Input parameter.

DY is COMPLEX

DY is an array, dimension (N). Workspace to hold the intermediate solution.

Y_TAIL Input parameter.

Y_TAIL is COMPLEX

Y_TAIL is an array, dimension (N). Workspace to hold the trailing bits of the intermediate solution.

RCOND Input parameter.

RCOND is REAL

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

ITHRESH Input parameter.

ITHRESH is INTEGER

The maximum number of residual computations allowed for refinement. The default is 10. For 'aggressive' set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in ERR_BNDS_NORM and ERR_BNDS_COMP may no longer be trustworthy.

RTHRESH Input parameter.

RTHRESH is REAL

Determines when to stop refinement if the error estimate stops decreasing. Refinement will stop when the next solution no longer satisfies norm(dx_{i+1}) < RTHRESH * norm(dx_i) where norm(Z) is the infinity norm of Z. RTHRESH satisfies 0 < RTHRESH <= 1. The default value is 0.5. For 'aggressive' set to 0.9 to permit convergence on extremely ill-conditioned matrices. See LAWN 165 for more details.

DZ_UB Input parameter.

DZ_UB is REAL

Determines when to start considering componentwise convergence. Componentwise convergence is only considered after each component of the solution Y is stable, which we definte as the relative change in each component being less than DZ_UB. The default value is 0.25, requiring the first bit to be stable. See LAWN 165 for more details.

IGNORE_CWISE Input parameter.

IGNORE_CWISE is LOGICAL

If .TRUE. then ignore componentwise convergence. Default value is .FALSE..

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. < 0: if INFO = -i, the ith argument to CLA_HERFSX_EXTENDED had an illegal value

Related Information

For this routine in other precisions, please see *zla_herfsx_extended*.

4.17.38 cla_herpvgrw

CLA_HERPVGRW computes the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If this is much less than 1, the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, estimated condition numbers, and error bounds could be unreliable.

Syntax

Fortran specification:

```
use armpl_library

real function cla_herpvgrw(UPLO, N, INFO, A, LDA, AF, LDAF, IPIV, WORK)
```

C specification:

```
#include "armpl.h"

float cla_herpvgrw_(const char *uplo, const armpl_int_t *n,
                    const armpl_int_t *info, const armpl_singlecomplex_t *a,
                    const armpl_int_t *lda, const armpl_singlecomplex_t *af,
                    const armpl_int_t *ldaf, const armpl_int_t *ipiv,
                    float *work, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

INFO Input parameter.

INFO is INTEGER

The value of INFO returned from SSYTRF, .i.e., the pivot in column INFO is exactly 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX

AF is an array, dimension (LDAF, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CHETRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CHETRF.

WORK Input parameter.

WORK is REAL

WORK is an array, dimension (2*N) .

Related Information

For this routine in other precisions, please see *zla_herpvgrw*.

4.17.39 cla_lin_berr

```
CLA_LIN_BERR computes componentwise relative backward error from
   the formula
       max(i) ( abs(R(i)) / ( abs(op(A_s))*abs(Y) + abs(B_s) )(i) )
   where abs(Z) is the componentwise absolute value of the matrix
   or vector Z.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine cla_lin_berr(N, NZ, NRHS, RES, AYB, BERR)
```

C specification:

```
#include "armpl.h"

void cla_lin_berr_(const armpl_int_t *n, const armpl_int_t *nz,
                   const armpl_int_t *nrhs, const armpl_singlecomplex_t *res,
                   const float *ayb, float *berr);
```

Parameters

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NZ Input parameter.

NZ is INTEGER

We add (NZ+1)*SLAMCH('Safe minimum') to R(i) in the numerator to guard against spuriously zero residuals. Default value is N.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices AYB, RES, and BERR. NRHS >= 0.

RES Input parameter.

RES is COMPLEX

RES is an array, dimension (N, NRHS). The residual matrix, i.e., the matrix R in the relative backward error formula above.

AYB Input parameter.

AYB is REAL

AYB is an array, dimension (N, NRHS). The denominator in the relative backward error formula above, i.e., the matrix abs(op(A_s))*abs(Y) + abs(B_s). The matrices A, Y, and B are from iterative refinement (see cla_gerfsx_extended.f).

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error from the formula above.

Related Information

For this routine in other precisions, please see *dla_lin_berr*, *sla_lin_berr* and *zla_lin_berr*.

4.17.40 cla_porcond_c

```
CLA_PORCOND_C Computes the infinity norm condition number of
   op(A) * inv(diag(C)) where C is a REAL vector
```

Syntax

Fortran specification:

```
use armpl_library

real function cla_porcond_c(UPLO, N, A, LDA, AF, LDAF, C, CAPPLY, INFO, WORK,
                            RWORK)
```

C specification:

```
#include "armpl.h"

float cla_porcond_c_(const char *uplo, const armpl_int_t *n,
                     const armpl_singlecomplex_t *a, const armpl_int_t *lda,
                     const armpl_singlecomplex_t *af, const armpl_int_t *ldaf,
                     const float *c, const armpl_int_t *capply,
                     armpl_int_t *info, const armpl_singlecomplex_t *work,
                     const float *rwork, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX

AF is an array, dimension (LDAF, N). The triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H , as computed by CPOTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

C Input parameter.

C is REAL

C is an array, dimension (N). The vector C in the formula op(A) * inv(diag(C)).

CAPPLY Input parameter.

CAPPLY is LOGICAL

If .TRUE. then access the vector C in the formula above.

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. i > 0: The ith argument is invalid.

WORK Input parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N).. Workspace.

RWORK Input parameter.

RWORK is REAL

RWORK is an array, dimension (N).. Workspace.

Related Information

For this routine in other precisions, please see *zla_porcond_c*.

4.17.41 cla_porcond_x

```
CLA_PORCOND_X Computes the infinity norm condition number of
   op(A) * diag(X) where X is a COMPLEX vector.
```

Syntax

Fortran specification:

```
use armpl_library

real function cla_porcond_x(UPLO, N, A, LDA, AF, LDAF, X, INFO, WORK, RWORK)
```

C specification:

```
#include "armpl.h"

float cla_porcond_x_(const char *uplo, const armpl_int_t *n,
                     const armpl_singlecomplex_t *a, const armpl_int_t *lda,
                     const armpl_singlecomplex_t *af, const armpl_int_t *ldaf,
                     const armpl_singlecomplex_t *x, armpl_int_t *info,
                     const armpl_singlecomplex_t *work, const float *rwork,
                     ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX

AF is an array, dimension (LDAF, N). The triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H , as computed by CPOTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

X Input parameter.

X is COMPLEX

X is an array, dimension (N). The vector X in the formula op(A) * diag(X).

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. i > 0: The ith argument is invalid.

WORK Input parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N).. Workspace.

RWORK Input parameter.

RWORK is REAL

RWORK is an array, dimension (N).. Workspace.

Related Information

For this routine in other precisions, please see *zla_porcond_x*.

4.17.42 cla_porfsx_extended

cla_porfsx_extended improves the computed solution to a system of linear equations by performing extra-precise iterative refinement and provides error bounds and backward error estimates for the solution. This subroutine is called by CPORFSX to perform iterative refinement. In addition to normwise error bound, the code provides maximum componentwise error bound if possible. See comments for ERR_BNDS_NORM and ERR_BNDS_COMP for details of the error bounds. Note that this subroutine is only resonsible for setting the second fields of ERR_BNDS_NORM and ERR_BNDS_COMP.

Syntax

Fortran specification:

```
use armpl_library

subroutine cla_porfsx_extended(PREC_TYPE, UPLO, N, NRHS, A, LDA, AF, LDAF,
                               COLEQU, C, B, LDB, Y, LDY, BERR_OUT, N_NORMS,
                               ERR_BNDS_NORM, ERR_BNDS_COMP, RES, AYB, DY,
                               Y_TAIL, RCOND, ITHRESH, RTHRESH, DZ_UB,
                               IGNORE_CWISE, INFO)
```

C specification:

```
#include "armpl.h"

void cla_porfsx_extended_(const armpl_int_t *prec_type, const char *uplo,
                          const armpl_int_t *n, const armpl_int_t *nrhs,
                          const armpl_singlecomplex_t *a,
                          const armpl_int_t *lda,
                          const armpl_singlecomplex_t *af,
                          const armpl_int_t *ldaf, const armpl_int_t *colequ,
                          const float *c, const armpl_singlecomplex_t *b,
                          const armpl_int_t *ldb, armpl_singlecomplex_t *y,
                          const armpl_int_t *ldy, float *berr_out,
                          const armpl_int_t *n_norms, float *err_bnds_norm,
                          float *err_bnds_comp, armpl_singlecomplex_t *res,
                          float *ayb, armpl_singlecomplex_t *dy,
                          armpl_singlecomplex_t *y_tail, const float *rcond,
                          const armpl_int_t *ithresh, const float *rthresh,
                          const float *dz_ub, const armpl_int_t *ignore_cwise,
                          armpl_int_t *info, ... );
```

Parameters

PREC_TYPE Input parameter.

PREC_TYPE is INTEGER

Specifies the intermediate precision to be used in refinement. The value is defined by ILAPREC(P) where P is a CHARACTER and P = 'S': Single = 'D': Double = 'I': Indigenous = 'X', 'E': Extra

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right-hand-sides, i.e., the number of columns of the matrix B.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX

AF is an array, dimension (LDAF, N). The triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , as computed by CPOTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

COLEQU Input parameter.

COLEQU is LOGICAL

If .TRUE. then column equilibration was done to A before calling this routine. This is needed to compute the solution and error bounds correctly.

C Input parameter.

C is REAL

C is an array, dimension (N). The column scale factors for A. If COLEQU = .FALSE., C is not accessed. If C is input, each element of C should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). The right-hand-side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Y Input and output parameter.

Y is COMPLEX

Y is an array, dimension (LDY, NRHS). On entry, the solution matrix X, as computed by CPOTRS. On exit, the improved solution matrix Y.

LDY Input parameter.

LDY is INTEGER

The leading dimension of the array Y. LDY >= max(1, N).

BERR_OUT Output parameter.

BERR_OUT is REAL

BERR_OUT is an array, dimension (NRHS). On exit, BERR_OUT(j) contains the componentwise relative backward error for right-hand-side j from the formula max(i) (abs(RES(i)) / (abs(op(A_s))*abs(Y) + abs(B_s))(i)) where abs(Z) is the componentwise absolute value of the matrix or vector Z. This is computed by CLA_LIN_BERR.

N_NORMS Input parameter.

N_NORMS is INTEGER

Determines which error bounds to return (see ERR_BNDS_NORM and ERR_BNDS_COMP). If N_NORMS >= 1 return normwise error bounds. If N_NORMS >= 2 return componentwise error bounds.

ERR_BNDS_NORM Input and output parameter.

ERR_BNDS_NORM is REAL

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 “Trust/don’t trust” boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch(‘Epsilon’).

err = 2 “Guaranteed” error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch(‘Epsilon’). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch(‘Epsilon’) to determine if the error estimate is “guaranteed”. These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

This subroutine is only responsible for setting the second field above. See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Input and output parameter.

ERR_BNDS_COMP is REAL

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:,N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 “Trust/don’t trust” boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch(‘Epsilon’).

err = 2 “Guaranteed” error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch(‘Epsilon’). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch(‘Epsilon’) to determine if the error estimate is “guaranteed”. These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

This subroutine is only responsible for setting the second field above. See Lapack Working Note 165 for further details and extra cautions.

RES Input parameter.

RES is COMPLEX

RES is an array, dimension (N). Workspace to hold the intermediate residual.

AYB Input parameter.

AYB is REAL

AYB is an array, dimension (N). Workspace.

DY Input parameter.

DY is COMPLEX

DY is an array, dimension (N). Workspace to hold the intermediate solution.

Y_TAIL Input parameter.

Y_TAIL is COMPLEX

Y_TAIL is an array, dimension (N). Workspace to hold the trailing bits of the intermediate solution.

RCOND Input parameter.

RCOND is REAL

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

ITHRESH Input parameter.

ITHRESH is INTEGER

The maximum number of residual computations allowed for refinement. The default is 10. For 'aggressive' set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in ERR_BNDS_NORM and ERR_BNDS_COMP may no longer be trustworthy.

RTHRESH Input parameter.

RTHRESH is REAL

Determines when to stop refinement if the error estimate stops decreasing. Refinement will stop when the next solution no longer satisfies norm(dx_{i+1}) < RTHRESH * norm(dx_i) where norm(Z) is the infinity norm of Z. RTHRESH satisfies 0 < RTHRESH <= 1. The default value is 0.5. For 'aggressive' set to 0.9 to permit convergence on extremely ill-conditioned matrices. See LAWN 165 for more details.

DZ_UB Input parameter.

DZ_UB is REAL

Determines when to start considering componentwise convergence. Componentwise convergence is only considered after each component of the solution Y is stable, which we definte as the relative change in each component being less than DZ_UB. The default value is 0.25, requiring the first bit to be stable. See LAWN 165 for more details.

IGNORE_CWISE Input parameter.

IGNORE_CWISE is LOGICAL

If .TRUE. then ignore componentwise convergence. Default value is .FALSE..

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. < 0: if INFO = -i, the ith argument to CPOTRS had an illegal value

Related Information

For this routine in other precisions, please see *dla_porfsx_extended*, *sla_porfsx_extended* and *zla_porfsx_extended*.

4.17.43 cla_porpvgrw

CLA_PORPVGRW computes the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If this is much less than 1, the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, estimated condition numbers, and error bounds could be unreliable.

Syntax

Fortran specification:

```
use armpl_library

real function cla_porpvgrw(UPLO, NCOLS, A, LDA, AF, LDAF, WORK)
```

C specification:

```
#include "armpl.h"

float cla_porpvgrw_(const char *uplo, const armpl_int_t *ncols,
                    const armpl_singlecomplex_t *a, const armpl_int_t *lda,
                    const armpl_singlecomplex_t *af, const armpl_int_t *ldaf,
                    float *work, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

NCOLS Input parameter.

NCOLS is INTEGER

The number of columns of the matrix A. NCOLS >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA,N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1,N).

AF Input parameter.

AF is COMPLEX

AF is an array, dimension (LDAF,N). The triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , as computed by CPOTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1,N).

WORK Input parameter.

WORK is REAL

WORK is an array, dimension (2*N) .

Related Information

For this routine in other precisions, please see *dla_porpvgrw*, *sla_porpvgrw* and *zla_porpvgrw*.

4.17.44 cla_syamv

CLA_SYAMV performs the matrix-vector operation

```
y := alpha*abs(A)*abs(x) + beta*abs(y),
```

where alpha and beta are scalars, x and y are vectors and A is an n by n symmetric matrix.

This function is primarily used in calculating error bounds. To protect against underflow during evaluation, components in the resulting vector are perturbed away from zero by (N+1) times the underflow threshold. To prevent unnecessarily large errors for block-structure embedded in general matrices, "symbolically" zero components are not perturbed. A zero entry is considered "symbolic" if all multiplications involved in computing that entry have at least one zero multiplicand.

Syntax

Fortran specification:

```
use armpl_library

subroutine cla_syamv(UPLO, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
```

C specification:

```
#include "armpl.h"

void cla_syamv_(const armpl_int_t *uplo, const armpl_int_t *n,
                const float *alpha, const armpl_singlecomplex_t *a,
                const armpl_int_t *lda, const armpl_singlecomplex_t *x,
                const armpl_int_t *incx, const float *beta, float *y,
                const armpl_int_t *incy);
```

Parameters

UPLO Input parameter.

UPLO is INTEGER

On entry, UPLO specifies whether the upper or lower triangular part of the array A is to be referenced as follows:

UPLO = BLAS_UPPER Only the upper triangular part of A is to be referenced.

UPLO = BLAS_LOWER Only the lower triangular part of A is to be referenced.

Unchanged on exit.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix A. N must be at least zero. Unchanged on exit.

ALPHA Input parameter.

ALPHA is REAL .

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, n).. Before entry, the leading m by n part of the array A must contain the matrix of coefficients. Unchanged on exit.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, n). Unchanged on exit.

X Input parameter.

X is COMPLEX

X is an array, dimension. (1 + (n - 1)*abs(INCX)) Before entry, the incremented array X must contain the vector x. Unchanged on exit.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

BETA Input parameter.

BETA is REAL .

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input. Unchanged on exit.

Y Input and output parameter.

Y is REAL

Y is an array, dimension. (1 + (n - 1)*abs(INCY)) Before entry with BETA non-zero, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero. Unchanged on exit.

Related Information

For this routine in other precisions, please see *dla_syamv*, *sla_syamv* and *zla_syamv*.

4.17.45 cla_syrcond_c

```
CLA_SYRCOND_C Computes the infinity norm condition number of
   op(A) * inv(diag(C)) where C is a REAL vector.
```

Syntax

Fortran specification:

```
use armpl_library

real function cla_syrcond_c(UPLO, N, A, LDA, AF, LDAF, IPIV, C, CAPPLY, INFO,
                            WORK, RWORK)
```

C specification:

```
#include "armpl.h"

float cla_syrcond_c_(const char *uplo, const armpl_int_t *n,
                     const armpl_singlecomplex_t *a, const armpl_int_t *lda,
```

(continues on next page)

(continued from previous page)

```
                    const armpl_singlecomplex_t *af, const armpl_int_t *ldaf,
                    const armpl_int_t *ipiv, const float *c,
                    const armpl_int_t *capply, armpl_int_t *info,
                    const armpl_singlecomplex_t *work, const float *rwork,
                    ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX

AF is an array, dimension (LDAF, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CSYTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CSYTRF.

C Input parameter.

C is REAL

C is an array, dimension (N). The vector C in the formula op(A) * inv(diag(C)).

CAPPLY Input parameter.

CAPPLY is LOGICAL

If .TRUE. then access the vector C in the formula above.

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. i > 0: The ith argument is invalid.

WORK Input parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N).. Workspace.

RWORK Input parameter.

RWORK is REAL

RWORK is an array, dimension (N).. Workspace.

Related Information

For this routine in other precisions, please see *zla_syrcond_c*.

4.17.46 cla_syrcond_x

```
CLA_SYRCOND_X Computes the infinity norm condition number of
   op(A) * diag(X) where X is a COMPLEX vector.
```

Syntax

Fortran specification:

```
use armpl_library

real function cla_syrcond_x(UPLO, N, A, LDA, AF, LDAF, IPIV, X, INFO, WORK,
                            RWORK)
```

C specification:

```
#include "armpl.h"

float cla_syrcond_x_(const char *uplo, const armpl_int_t *n,
                     const armpl_singlecomplex_t *a, const armpl_int_t *lda,
                     const armpl_singlecomplex_t *af, const armpl_int_t *ldaf,
                     const armpl_int_t *ipiv, const armpl_singlecomplex_t *x,
                     armpl_int_t *info, const armpl_singlecomplex_t *work,
                     const float *rwork, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX

AF is an array, dimension (LDAF, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CSYTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CSYTRF.

X Input parameter.

X is COMPLEX

X is an array, dimension (N). The vector X in the formula op(A) * diag(X).

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. i > 0: The ith argument is invalid.

WORK Input parameter.

WORK is COMPLEX

WORK is an array, dimension (2*N).. Workspace.

RWORK Input parameter.

RWORK is REAL

RWORK is an array, dimension (N).. Workspace.

Related Information

For this routine in other precisions, please see *zla_syrcond_x*.

4.17.47 cla_syrfsx_extended

cla_syrfsx_extended improves the computed solution to a system of linear equations by performing extra-precise iterative refinement and provides error bounds and backward error estimates for the solution. This subroutine is called by CSYRFSX to perform iterative refinement. In addition to normwise error bound, the code provides maximum componentwise error bound if possible. See comments for ERR_BNDS_NORM and ERR_BNDS_COMP for details of the error bounds. Note that this subroutine is only resonsible for setting the second fields of ERR_BNDS_NORM and ERR_BNDS_COMP.

Syntax

Fortran specification:

```
use armpl_library

subroutine cla_syrfsx_extended(PREC_TYPE, UPLO, N, NRHS, A, LDA, AF, LDAF,
                               IPIV, COLEQU, C, B, LDB, Y, LDY, BERR_OUT,
                               N_NORMS, ERR_BNDS_NORM, ERR_BNDS_COMP, RES,
                               AYB, DY, Y_TAIL, RCOND, ITHRESH, RTHRESH,
                               DZ_UB, IGNORE_CWISE, INFO)
```

C specification:

```
#include "armpl.h"

void cla_syrfsx_extended_(const armpl_int_t *prec_type, const char *uplo,
                          const armpl_int_t *n, const armpl_int_t *nrhs,
                          const armpl_singlecomplex_t *a,
                          const armpl_int_t *lda,
                          const armpl_singlecomplex_t *af,
                          const armpl_int_t *ldaf, const armpl_int_t *ipiv,
                          const armpl_int_t *colequ, const float *c,
                          const armpl_singlecomplex_t *b,
                          const armpl_int_t *ldb, armpl_singlecomplex_t *y,
                          const armpl_int_t *ldy, float *berr_out,
                          const armpl_int_t *n_norms, float *err_bnds_norm,
                          float *err_bnds_comp, armpl_singlecomplex_t *res,
                          float *ayb, armpl_singlecomplex_t *dy,
                          armpl_singlecomplex_t *y_tail, const float *rcond,
                          const armpl_int_t *ithresh, const float *rthresh,
                          const float *dz_ub, const armpl_int_t *ignore_cwise,
                          armpl_int_t *info, ... );
```

Parameters

PREC_TYPE Input parameter.

PREC_TYPE is INTEGER

Specifies the intermediate precision to be used in refinement. The value is defined by ILAPREC(P) where P is a CHARACTER and P = ‘S’: Single = ‘D’: Double = ‘I’: Indigenous = ‘X’, ‘E’: Extra

UPLO Input parameter.

UPLO is CHARACTER*1

= ‘U’: Upper triangle of A is stored; = ‘L’: Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right-hand-sides, i.e., the number of columns of the matrix B.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX

AF is an array, dimension (LDAF, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CSYTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CSYTRF.

COLEQU Input parameter.

COLEQU is LOGICAL

If .TRUE. then column equilibration was done to A before calling this routine. This is needed to compute the solution and error bounds correctly.

C Input parameter.

C is REAL

C is an array, dimension (N). The column scale factors for A. If COLEQU = .FALSE., C is not accessed. If C is input, each element of C should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). The right-hand-side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Y Input and output parameter.

Y is COMPLEX

Y is an array, dimension (LDY, NRHS). On entry, the solution matrix X, as computed by CSYTRS. On exit, the improved solution matrix Y.

LDY Input parameter.

LDY is INTEGER

The leading dimension of the array Y. LDY >= max(1, N).

BERR_OUT Output parameter.

BERR_OUT is REAL

BERR_OUT is an array, dimension (NRHS). On exit, BERR_OUT(j) contains the componentwise relative backward error for right-hand-side j from the formula max(i) (abs(RES(i)) / (abs(op(A_s))*abs(Y) + abs(B_s))(i)) where abs(Z) is the componentwise absolute value of the matrix or vector Z. This is computed by CLA_LIN_BERR.

N_NORMS Input parameter.

N_NORMS is INTEGER

Determines which error bounds to return (see ERR_BNDS_NORM and ERR_BNDS_COMP). If N_NORMS >= 1 return normwise error bounds. If N_NORMS >= 2 return componentwise error bounds.

ERR_BNDS_NORM Input and output parameter.

ERR_BNDS_NORM is REAL

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

This subroutine is only responsible for setting the second field above. See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Input and output parameter.

ERR_BNDS_COMP is REAL

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:,N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix

Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

This subroutine is only responsible for setting the second field above. See Lapack Working Note 165 for further details and extra cautions.

RES Input parameter.

RES is COMPLEX

RES is an array, dimension (N). Workspace to hold the intermediate residual.

AYB Input parameter.

AYB is REAL

AYB is an array, dimension (N). Workspace.

DY Input parameter.

DY is COMPLEX

DY is an array, dimension (N). Workspace to hold the intermediate solution.

Y_TAIL Input parameter.

Y_TAIL is COMPLEX

Y_TAIL is an array, dimension (N). Workspace to hold the trailing bits of the intermediate solution.

RCOND Input parameter.

RCOND is REAL

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

ITHRESH Input parameter.

ITHRESH is INTEGER

The maximum number of residual computations allowed for refinement. The default is 10. For 'aggressive' set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in ERR_BNDS_NORM and ERR_BNDS_COMP may no longer be trustworthy.

RTHRESH Input parameter.

RTHRESH is REAL

Determines when to stop refinement if the error estimate stops decreasing. Refinement will stop when the next solution no longer satisfies norm(dx_{i+1}) < RTHRESH * norm(dx_i) where norm(Z) is the infinity norm of Z. RTHRESH satisfies 0 < RTHRESH <= 1. The default value is 0.5. For 'aggressive' set to 0.9 to permit convergence on extremely ill-conditioned matrices. See LAWN 165 for more details.

DZ_UB Input parameter.

DZ_UB is REAL

Determines when to start considering componentwise convergence. Componentwise convergence is only considered after each component of the solution Y is stable, which we definte as the relative change in each component being less than DZ_UB. The default value is 0.25, requiring the first bit to be stable. See LAWN 165 for more details.

IGNORE_CWISE Input parameter.

IGNORE_CWISE is LOGICAL

If .TRUE. then ignore componentwise convergence. Default value is .FALSE..

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. < 0: if INFO = -i, the ith argument to CLA_SYRFSX_EXTENDED had an illegal value

Related Information

For this routine in other precisions, please see *dla_syrfsx_extended*, *sla_syrfsx_extended* and *zla_syrfsx_extended*.

4.17.48 cla_syrpvgrw

CLA_SYRPVGRW computes the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If this is much less than 1, the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, estimated condition numbers, and error bounds could be unreliable.

Syntax

Fortran specification:

```
use armpl_library

real function cla_syrpvgrw(UPLO, N, INFO, A, LDA, AF, LDAF, IPIV, WORK)
```

C specification:

```
#include "armpl.h"

float cla_syrpvgrw_(const char *uplo, const armpl_int_t *n,
                    const armpl_int_t *info, const armpl_singlecomplex_t *a,
                    const armpl_int_t *lda, const armpl_singlecomplex_t *af,
                    const armpl_int_t *ldaf, const armpl_int_t *ipiv,
                    float *work, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

INFO Input parameter.

INFO is INTEGER

The value of INFO returned from CSYTRF, .i.e., the pivot in column INFO is exactly 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX

AF is an array, dimension (LDAF, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CSYTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CSYTRF.

WORK Input parameter.

WORK is REAL

WORK is an array, dimension (2*N) .

Related Information

For this routine in other precisions, please see *dla_syrpvgrw*, *sla_syrpvgrw* and *zla_syrpvgrw*.

4.17.49 cla_wwaddw

```
CLA_WWADDW adds a vector W into a doubled-single vector (X, Y).

   This works for all extant IBM's hex and binary floating point
   arithmetics, but not for decimal.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine cla_wwaddw(N, X, Y, W)
```

C specification:

```
#include "armpl.h"

void cla_wwaddw_(const armpl_int_t *n, armpl_singlecomplex_t *x,
                 armpl_singlecomplex_t *y, const armpl_singlecomplex_t *w);
```

Parameters

N Input parameter.

N is INTEGER

The length of vectors X, Y, and W.

X Input and output parameter.

X is COMPLEX

X is an array, dimension (N). The first part of the doubled-single accumulation vector.

Y Input and output parameter.

Y is COMPLEX

Y is an array, dimension (N). The second part of the doubled-single accumulation vector.

W Input parameter.

W is COMPLEX

W is an array, dimension (N). The vector to be added.

Related Information

For this routine in other precisions, please see *dla_wwaddw*, *sla_wwaddw* and *zla_wwaddw*.

4.17.50 clabrd

clabrd reduces the first NB rows and columns of a complex general m by n matrix A to upper or lower real bidiagonal form by a unitary transformation $Q^H * A * P$, and returns the matrices X and Y which are needed to apply the transformation to the unreduced part of A.

If m >= n, A is reduced to upper bidiagonal form; if m < n, to lower bidiagonal form.

This is an auxiliary routine called by CGEBRD

Syntax

Fortran specification:

```
use armpl_library

subroutine clabrd(M, N, NB, A, LDA, D, E, TAUQ, TAUP, X, LDX, Y, LDY)
```

C specification:

```
#include "armpl.h"

void clabrd_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *nb, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, float *d, float *e,
             armpl_singlecomplex_t *tauq, armpl_singlecomplex_t *taup,
             armpl_singlecomplex_t *x, const armpl_int_t *ldx,
             armpl_singlecomplex_t *y, const armpl_int_t *ldy);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows in the matrix A.

N Input parameter.

N is INTEGER

The number of columns in the matrix A.

NB Input parameter.

NB is INTEGER

The number of leading rows and columns of A to be reduced.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the m by n general matrix to be reduced. On exit, the first NB rows and columns of the matrix are overwritten; the rest of the array is unchanged. If m >= n, elements on and below the diagonal in the first NB columns, with the array TAUQ, represent the unitary matrix Q as a product of elementary reflectors; and elements above the diagonal in the first NB rows, with the array TAUP, represent the unitary matrix P as a product of elementary reflectors. If m < n, elements below the diagonal in the first NB columns, with the array TAUQ, represent the unitary matrix Q as a product of elementary reflectors, and elements on and above the diagonal in the first NB rows, with the array TAUP, represent the unitary matrix P as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

D Output parameter.

D is REAL

D is an array, dimension (NB). The diagonal elements of the first NB rows and columns of the reduced matrix. D(i) = A(i,i).

E Output parameter.

E is REAL

E is an array, dimension (NB). The off-diagonal elements of the first NB rows and columns of the reduced matrix.

TAUQ Output parameter.

TAUQ is COMPLEX

TAUQ is an array, dimension (NB). The scalar factors of the elementary reflectors which represent the unitary matrix Q. See Further Details.

TAUP Output parameter.

TAUP is COMPLEX

TAUP is an array, dimension (NB). The scalar factors of the elementary reflectors which represent the unitary matrix P. See Further Details.

X Output parameter.

X is COMPLEX

X is an array, dimension (LDX, NB). The m-by-nb matrix X required to update the unreduced part of A.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, M).

Y Output parameter.

Y is COMPLEX

Y is an array, dimension (LDY, NB). The n-by-nb matrix Y required to update the unreduced part of A.

LDY Input parameter.

LDY is INTEGER

The leading dimension of the array Y. LDY >= max(1, N).

Related Information

For this routine in other precisions, please see *dlabrd*, *slabrd* and *zlabrd*.

4.17.51 clacgv

clacgv conjugates a complex vector of length N.

Syntax

Fortran specification:

```
use armpl_library

subroutine clacgv(N, X, INCX)
```

C specification:

```
#include "armpl.h"

void clacgv_(const armpl_int_t *n, armpl_singlecomplex_t *x,
             const armpl_int_t *incx);
```

Parameters

N Input parameter.

N is INTEGER

The length of the vector X. N >= 0.

X Input and output parameter.

X is COMPLEX

X is an array, dimension. (1+(N-1)*abs(INCX)) On entry, the vector of length N to be conjugated. On exit, X is overwritten with conjg(X).

INCX Input parameter.

INCX is INTEGER

The spacing between successive elements of X.

Related Information

For this routine in other precisions, please see *zlacgv*. It also exists with a native C interface as *LAPACKE_clacgv*.

4.17.52 clacn2

clacn2 estimates the 1-norm of a square, complex matrix A. Reverse communication is used for evaluating matrix-vector products.

Syntax

Fortran specification:

```
use armpl_library

subroutine clacn2(N, V, X, EST, KASE, ISAVE)
```

C specification:

```
#include "armpl.h"

void clacn2_(const armpl_int_t *n, armpl_singlecomplex_t *v,
             armpl_singlecomplex_t *x, float *est, armpl_int_t *kase,
             armpl_int_t *isave);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix. N >= 1.

V Output parameter.

V is COMPLEX

V is an array, dimension (N). On the final return, V = A*W, where EST = norm(V)/norm(W) (W is not returned).

X Input and output parameter.

X is COMPLEX

X is an array, dimension (N). On an intermediate return, X should be overwritten by A * X, if KASE=1, A^H * X, if KASE=2, where A^H is the conjugate transpose of A, and CLACN2 must be re-called with all the other parameters unchanged.

EST Input and output parameter.

EST is REAL

On entry with KASE = 1 or 2 and ISAVE(1) = 3, EST should be unchanged from the previous call to CLACN2. On exit, EST is an estimate (a lower bound) for norm(A).

KASE Input and output parameter.

KASE is INTEGER

On the initial call to CLACN2, KASE should be 0. On an intermediate return, KASE will be 1 or 2, indicating whether X should be overwritten by A * X or A^H * X. On the final return from CLACN2, KASE will again be 0.

ISAVE Input and output parameter.

ISAVE is INTEGER array, dimension (3)

ISAVE is used to save variables between calls to SLACN2

Related Information

For this routine in other precisions, please see *dlacn2*, *slacn2* and *zlacn2*. It also exists with a native C interface as *LAPACKE_clacn2*.

4.17.53 clacon

`clacon` estimates the 1-norm of a square, complex matrix A. Reverse communication is used for evaluating matrix-vector products.

Syntax

Fortran specification:

```
use armpl_library

subroutine clacon(N, V, X, EST, KASE)
```

C specification:

```
#include "armpl.h"

void clacon_(const armpl_int_t *n, armpl_singlecomplex_t *v,
             armpl_singlecomplex_t *x, float *est, armpl_int_t *kase);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix. N >= 1.

V Output parameter.

V is COMPLEX

V is an array, dimension (N). On the final return, V = A*W, where EST = norm(V)/norm(W) (W is not returned).

X Input and output parameter.

X is COMPLEX

X is an array, dimension (N). On an intermediate return, X should be overwritten by A * X, if KASE=1, A^H * X, if KASE=2, where A^H is the conjugate transpose of A, and CLACON must be re-called with all the other parameters unchanged.

EST Input and output parameter.

EST is REAL

On entry with KASE = 1 or 2 and JUMP = 3, EST should be unchanged from the previous call to CLACON. On exit, EST is an estimate (a lower bound) for norm(A).

KASE Input and output parameter.

KASE is INTEGER

On the initial call to CLACON, KASE should be 0. On an intermediate return, KASE will be 1 or 2, indicating whether X should be overwritten by A * X or A^H * X. On the final return from CLACON, KASE will again be 0.

Related Information

For this routine in other precisions, please see *dlacon*, *slacon* and *zlacon*.

4.17.54 clacp2

`clacp2` copies all or part of a real two-dimensional matrix A to a complex matrix B.

Syntax

Fortran specification:

```
use armpl_library

subroutine clacp2(UPLO, M, N, A, LDA, B, LDB)
```

C specification:

```
#include "armpl.h"

void clacp2_(const char *uplo, const armpl_int_t *m, const armpl_int_t *n,
             const float *a, const armpl_int_t *lda, armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies the part of the matrix A to be copied to B. = 'U': Upper triangular part = 'L': Lower triangular part Otherwise: All of the matrix A

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). The m by n matrix A. If UPLO = 'U', only the upper trapezium is accessed; if UPLO = 'L', only the lower trapezium is accessed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Output parameter.

B is COMPLEX

B is an array, dimension (LDB, N). On exit, B = A in the locations specified by UPLO.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M).

Related Information

For this routine in other precisions, please see *zlacp2*. It also exists with a native C interface as *LAPACKE_clacp2*.

4.17.55 clacpy

clacpy copies all or part of a two-dimensional matrix A to another matrix B.

Syntax

Fortran specification:

```
use armpl_library

subroutine clacpy(UPLO, M, N, A, LDA, B, LDB)
```

C specification:

```
#include "armpl.h"

void clacpy_(const char *uplo, const armpl_int_t *m, const armpl_int_t *n,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda,
             armpl_singlecomplex_t *b, const armpl_int_t *ldb, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies the part of the matrix A to be copied to B. = 'U': Upper triangular part = 'L': Lower triangular part Otherwise: All of the matrix A

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The m by n matrix A. If UPLO = 'U', only the upper trapezium is accessed; if UPLO = 'L', only the lower trapezium is accessed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Output parameter.

B is COMPLEX

B is an array, dimension (LDB, N). On exit, B = A in the locations specified by UPLO.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M).

Related Information

For this routine in other precisions, please see *dlacpy*, *slacpy* and *zlacpy*. It also exists with a native C interface as *LAPACKE_clacpy*.

4.17.56 clacrm

clacrm performs a very simple matrix-matrix multiplication:

```
C := A * B,
```

where A is M by N and complex; B is N by N and real; C is M by N and complex.

Syntax

Fortran specification:

```
use armpl_library

subroutine clacrm(M, N, A, LDA, B, LDB, C, LDC, RWORK)
```

C specification:

```
#include "armpl.h"

void clacrm_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const float *b, const armpl_int_t *ldb, armpl_singlecomplex_t *c,
             const armpl_int_t *ldc, float *rwork);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A and of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns and rows of the matrix B and the number of columns of the matrix C. N >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, A contains the M by N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >=max(1, M).

B Input parameter.

B is REAL

B is an array, dimension (LDB, N). On entry, B contains the N by N matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >=max(1, N).

C Output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). On exit, C contains the M by N matrix C.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >=max(1, N).

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (2*M*N) .

Related Information

For this routine in other precisions, please see *zlacrm*. It also exists with a native C interface as *LAPACKE_clacrm*.

4.17.57 clacrt

clacrt performs the operation

```
(  c  s )( x )  ==> ( x )
( -s  c )( y )      ( y )
```

where c and s are complex and the vectors x and y are complex.

Syntax

Fortran specification:

```
use armpl_library

subroutine clacrt(N, CX, INCX, CY, INCY, C, S)
```

C specification:

```
#include "armpl.h"

void clacrt_(const armpl_int_t *n, armpl_singlecomplex_t *cx,
             const armpl_int_t *incx, armpl_singlecomplex_t *cy,
             const armpl_int_t *incy, const armpl_singlecomplex_t *c,
             const armpl_singlecomplex_t *s);
```

Parameters

N Input parameter.

N is INTEGER

The number of elements in the vectors CX and CY.

CX Input and output parameter.

CX is COMPLEX

CX is an array, dimension (N). On input, the vector x. On output, CX is overwritten with c*x + s*y.

INCX Input parameter.

INCX is INTEGER

The increment between successive values of CX. INCX <> 0.

CY Input and output parameter.

CY is COMPLEX

CY is an array, dimension (N). On input, the vector y. On output, CY is overwritten with -s*x + c*y.

INCY Input parameter.

INCY is INTEGER

The increment between successive values of CY. INCY <> 0.

C Input parameter.

C is COMPLEX

S Input parameter.

S is COMPLEX

C and S define the matrix [C S]. [-S C]

Related Information

For this routine in other precisions, please see *zlacrt*.

4.17.58 cladiv

cladiv := X / Y, where X and Y are complex. The computation of X / Y will not overflow on an intermediary step unless the results overflows.

Syntax

Fortran specification:

```
use armpl_library

complex function cladiv(X, Y)
```

C specification:

```
#include "armpl.h"

SINGLECOMPLEX_RET_VALUE cladiv_(SINGLECOMPLEX_RET_PARAM const armpl_singlecomplex_
↪t *x,
                                const armpl_singlecomplex_t *y);
```

Parameters

X Input parameter.

X is COMPLEX

Y Input parameter.

Y is COMPLEX

The complex scalars X and Y.

Related Information

For this routine in other precisions, please see *dladiv*, *sladiv* and *zladiv*.

4.17.59 claed0

Using the divide and conquer method, claed0 computes all eigenvalues of a symmetric tridiagonal matrix which is one diagonal block of those from reducing a dense or band Hermitian matrix and corresponding eigenvectors of the dense or band matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine claed0(QSIZ, N, D, E, Q, LDQ, QSTORE, LDQS, RWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void claed0_(const armpl_int_t *qsiz, const armpl_int_t *n, float *d,
             float *e, armpl_singlecomplex_t *q, const armpl_int_t *ldq,
             armpl_singlecomplex_t *qstore, const armpl_int_t *ldqs,
             float *rwork, armpl_int_t *iwork, armpl_int_t *info);
```

Parameters

QSIZ Input parameter.

QSIZ is INTEGER

The dimension of the unitary matrix used to reduce the full matrix to tridiagonal form. QSIZ >= N if ICOMPQ = 1.

N Input parameter.

N is INTEGER

The dimension of the symmetric tridiagonal matrix. N >= 0.

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry, the diagonal elements of the tridiagonal matrix. On exit, the eigenvalues in ascending order.

E Input and output parameter.

E is REAL

E is an array, dimension (N-1). On entry, the off-diagonal elements of the tridiagonal matrix. On exit, E has been destroyed.

Q Input and output parameter.

Q is COMPLEX

Q is an array, dimension (LDQ, N). On entry, Q must contain an QSIZ x N matrix whose columns unitarily orthonormal. It is a part of the unitary matrix that reduces the full dense Hermitian matrix to a (reducible) symmetric tridiagonal matrix.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, N).

IWORK Output parameter.

IWORK is INTEGER array,

the dimension of IWORK must be at least 6 + 6* N + 5*N*lg N (lg(N) = smallest integer k such that 2^k >= N)

RWORK Output parameter.

RWORK is REAL

RWORK is an array,. dimension (1 + 3* N + 2*N*lg N + 3*N**2) (lg(N) = smallest integer k such that 2^k >= N)

QSTORE Output parameter.

QSTORE is COMPLEX

QSTORE is an array, dimension (LDQS, N). Used to store parts of the eigenvector matrix when the updating matrix multiplies take place.

LDQS Input parameter.

LDQS is INTEGER

The leading dimension of the array QSTORE. LDQS >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: The algorithm failed to compute an eigenvalue while working on the submatrix lying in rows and columns INFO/(N+1) through mod(INFO,N+1).

Related Information

For this routine in other precisions, please see *dlaed0*, *slaed0* and *zlaed0*.

4.17.60 claed7

claed7 computes the updated eigensystem of a diagonal matrix after modification by a rank-one symmetric matrix. This routine is used only for the eigenproblem which requires all eigenvalues and optionally eigenvectors of a dense or banded Hermitian matrix that has been reduced to tridiagonal form.

```
 T = Q(in) ( D(in) + RHO * Z*Z**H ) Q**H(in) = Q(out) * D(out) * Q**H(out)

where Z = Q**Hu, u is a vector of length N with ones in the
CUTPNT and CUTPNT + 1 th elements and zeros elsewhere.

 The eigenvectors of the original matrix are stored in Q, and the
 eigenvalues are in D.  The algorithm consists of three stages:

    The first stage consists of deflating the size of the problem
    when there are multiple eigenvalues or if there is a zero in
    the Z vector.  For each such occurrence the dimension of the
    secular equation problem is reduced by one.  This stage is
    performed by the routine SLAED2.

    The second stage consists of calculating the updated
    eigenvalues. This is done by finding the roots of the secular
    equation via the routine SLAED4 (as called by SLAED3).
    This routine also calculates the eigenvectors of the current
    problem.

    The final stage consists of computing the updated eigenvectors
    directly using the updated eigenvalues.  The eigenvectors for
    the current problem are multiplied with the eigenvectors from
    the overall problem.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine claed7(N, CUTPNT, QSIZ, TLVLS, CURLVL, CURPBM, D, Q, LDQ, RHO,
                  INDXQ, QSTORE, QPTR, PRMPTR, PERM, GIVPTR, GIVCOL, GIVNUM,
                  WORK, RWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void claed7_(const armpl_int_t *n, const armpl_int_t *cutpnt,
             const armpl_int_t *qsiz, const armpl_int_t *tlvls,
             const armpl_int_t *curlvl, const armpl_int_t *curpbm, float *d,
             armpl_singlecomplex_t *q, const armpl_int_t *ldq,
             const float *rho, armpl_int_t *indxq, float *qstore,
             armpl_int_t *qptr, const armpl_int_t *prmptr,
             const armpl_int_t *perm, const armpl_int_t *givptr,
             const armpl_int_t *givcol, const float *givnum,
             armpl_singlecomplex_t *work, float *rwork, armpl_int_t *iwork,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The dimension of the symmetric tridiagonal matrix. N >= 0.

CUTPNT Input parameter.

CUTPNT is INTEGER

Contains the location of the last eigenvalue in the leading sub-matrix. min(1, N) <= CUTPNT <= N.

QSIZ Input parameter.

QSIZ is INTEGER

The dimension of the unitary matrix used to reduce the full matrix to tridiagonal form. QSIZ >= N.

TLVLS Input parameter.

TLVLS is INTEGER

The total number of merging levels in the overall divide and conquer tree.

CURLVL Input parameter.

CURLVL is INTEGER

The current level in the overall merge routine, 0 <= curlvl <= tlvls.

CURPBM Input parameter.

CURPBM is INTEGER

The current problem in the current level in the overall merge routine (counting from upper left to lower right).

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry, the eigenvalues of the rank-1-perturbed matrix. On exit, the eigenvalues of the repaired matrix.

Q Input and output parameter.

Q is COMPLEX

Q is an array, dimension (LDQ, N). On entry, the eigenvectors of the rank-1-perturbed matrix. On exit, the eigenvectors of the repaired tridiagonal matrix.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, N).

RHO Input parameter.

RHO is REAL

Contains the subdiagonal element used to create the rank-1 modification.

INDXQ Output parameter.

INDXQ is INTEGER array, dimension (N)

This contains the permutation which will reintegrate the subproblem just solved back into sorted order, ie. D(INDXQ(I = 1, N)) will be in ascending order.

IWORK Output parameter.

IWORK is INTEGER array, dimension (4*N)

RWORK Output parameter.

RWORK is REAL

RWORK is an array,. dimension (3*N+2*QSIZ*N)

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (QSIZ*N) .

QSTORE Input and output parameter.

QSTORE is REAL

QSTORE is an array, dimension (N**2+1). Stores eigenvectors of submatrices encountered during divide and conquer, packed together. QPTR points to beginning of the submatrices.

QPTR Input and output parameter.

QPTR is INTEGER array, dimension (N+2)

List of indices pointing to beginning of submatrices stored in QSTORE. The submatrices are numbered starting at the bottom left of the divide and conquer tree, from left to right and bottom to top.

PRMPTR Input parameter.

PRMPTR is INTEGER array, dimension (N lg N)

Contains a list of pointers which indicate where in PERM a level's permutation is stored. PRMPTR(i+1) - PRMPTR(i) indicates the size of the permutation and also the size of the full, non-deflated problem.

PERM Input parameter.

PERM is INTEGER array, dimension (N lg N)

Contains the permutations (from deflation and sorting) to be applied to each eigenblock.

GIVPTR Input parameter.

GIVPTR is INTEGER array, dimension (N lg N)

Contains a list of pointers which indicate where in GIVCOL a level's Givens rotations are stored. GIVPTR(i+1) - GIVPTR(i) indicates the number of Givens rotations.

GIVCOL Input parameter.

GIVCOL is INTEGER array, dimension (2, N lg N)

Each pair of numbers indicates a pair of columns to take place in a Givens rotation.

GIVNUM Input parameter.

GIVNUM is REAL

GIVNUM is an array, dimension (2, N lg N). Each number indicates the S value to be used in the corresponding Givens rotation.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = 1, an eigenvalue did not converge

Related Information

For this routine in other precisions, please see *dlaed7*, *slaed7* and *zlaed7*.

4.17.61 claed8

claed8 merges the two sets of eigenvalues together into a single sorted set. Then it tries to deflate the size of the problem. There are two ways in which deflation can occur: when two or more eigenvalues are close together or if there is a tiny element in the Z vector. For each such occurrence the order of the related secular equation problem is reduced by one.

Syntax

Fortran specification:

```
use armpl_library

subroutine claed8(K, N, QSIZ, Q, LDQ, D, RHO, CUTPNT, Z, DLAMDA, Q2, LDQ2, W,
                  INDXP, INDX, INDXQ, PERM, GIVPTR, GIVCOL, GIVNUM, INFO)
```

C specification:

```
#include "armpl.h"

void claed8_(armpl_int_t *k, const armpl_int_t *n, const armpl_int_t *qsiz,
             armpl_singlecomplex_t *q, const armpl_int_t *ldq, float *d,
             float *rho, const armpl_int_t *cutpnt, const float *z,
             float *dlamda, armpl_singlecomplex_t *q2,
             const armpl_int_t *ldq2, float *w, armpl_int_t *indxp,
             armpl_int_t *indx, const armpl_int_t *indxq, armpl_int_t *perm,
             armpl_int_t *givptr, armpl_int_t *givcol, float *givnum,
             armpl_int_t *info);
```

Parameters

K Output parameter.

K is INTEGER

Contains the number of non-deflated eigenvalues. This is the order of the related secular equation.

N Input parameter.

N is INTEGER

The dimension of the symmetric tridiagonal matrix. N >= 0.

QSIZ Input parameter.

QSIZ is INTEGER

The dimension of the unitary matrix used to reduce the dense or band matrix to tridiagonal form. QSIZ >= N if ICOMPQ = 1.

Q Input and output parameter.

Q is COMPLEX

Q is an array, dimension (LDQ, N). On entry, Q contains the eigenvectors of the partially solved system which has been previously updated in matrix multiplies with other partially solved eigensystems. On exit, Q contains the trailing (N-K) updated eigenvectors (those which were deflated) in its last N-K columns.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, N).

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry, D contains the eigenvalues of the two submatrices to be combined. On exit, D contains the trailing (N-K) updated eigenvalues (those which were deflated) sorted into increasing order.

RHO Input and output parameter.

RHO is REAL

Contains the off diagonal element associated with the rank-1 cut which originally split the two submatrices which are now being recombined. RHO is modified during the computation to the value required by SLAED3.

CUTPNT Input parameter.

CUTPNT is INTEGER

Contains the location of the last eigenvalue in the leading sub-matrix. MIN(1, N) <= CUTPNT <= N.

Z Input parameter.

Z is REAL

Z is an array, dimension (N). On input this vector contains the updating vector (the last row of the first sub-eigenvector matrix and the first row of the second sub-eigenvector matrix). The contents of Z are destroyed during the updating process.

DLAMDA Output parameter.

DLAMDA is REAL

DLAMDA is an array, dimension (N). Contains a copy of the first K eigenvalues which will be used by SLAED3 to form the secular equation.

Q2 Output parameter.

Q2 is COMPLEX

Q2 is an array, dimension (LDQ2, N). If ICOMPQ = 0, Q2 is not referenced. Otherwise, Contains a copy of the first K eigenvectors which will be used by SLAED7 in a matrix multiply (SGEMM) to update the new eigenvectors.

LDQ2 Input parameter.

LDQ2 is INTEGER

The leading dimension of the array Q2. LDQ2 >= max(1, N).

W Output parameter.

W is REAL

W is an array, dimension (N). This will hold the first k values of the final deflation-altered z-vector and will be passed to SLAED3.

INDXP Output parameter.

INDXP is INTEGER array, dimension (N)

This will contain the permutation used to place deflated values of D at the end of the array. On output INDXP(1:K) points to the nondeflated D-values and INDXP(K+1:N) points to the deflated eigenvalues.

INDX Output parameter.

INDX is INTEGER array, dimension (N)

This will contain the permutation used to sort the contents of D into ascending order.

INDXQ Input parameter.

INDXQ is INTEGER array, dimension (N)

This contains the permutation which separately sorts the two sub-problems in D into ascending order. Note that elements in the second half of this permutation must first have CUTPNT added to their values in order to be accurate.

PERM Output parameter.

PERM is INTEGER array, dimension (N)

Contains the permutations (from deflation and sorting) to be applied to each eigenblock.

GIVPTR Output parameter.

GIVPTR is INTEGER

Contains the number of Givens rotations which took place in this subproblem.

GIVCOL Output parameter.

GIVCOL is INTEGER array, dimension (2, N)

Each pair of numbers indicates a pair of columns to take place in a Givens rotation.

GIVNUM Output parameter.

GIVNUM is REAL

GIVNUM is an array, dimension (2, N). Each number indicates the S value to be used in the corresponding Givens rotation.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *dlaed8*, *slaed8* and *zlaed8*.

4.17.62 claein

claein uses inverse iteration to find a right or left eigenvector corresponding to the eigenvalue W of a complex upper Hessenberg matrix H.

Syntax

Fortran specification:

```
use armpl_library

subroutine claein(RIGHTV, NOINIT, N, H, LDH, W, V, B, LDB, RWORK, EPS3,
                  SMLNUM, INFO)
```

C specification:

```
#include "armpl.h"

void claein_(const armpl_int_t *rightv, const armpl_int_t *noinit,
             const armpl_int_t *n, const armpl_singlecomplex_t *h,
             const armpl_int_t *ldh, const armpl_singlecomplex_t *w,
             armpl_singlecomplex_t *v, armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, float *rwork, const float *eps3,
             const float *smlnum, armpl_int_t *info);
```

Parameters

RIGHTV Input parameter.

RIGHTV is LOGICAL

= .TRUE. : compute right eigenvector; = .FALSE.: compute left eigenvector.

NOINIT Input parameter.

NOINIT is LOGICAL

= .TRUE. : no initial vector supplied in V = .FALSE.: initial vector supplied in V.

N Input parameter.

N is INTEGER

The order of the matrix H. N >= 0.

H Input parameter.

H is COMPLEX

H is an array, dimension (LDH, N). The upper Hessenberg matrix H.

LDH Input parameter.

LDH is INTEGER

The leading dimension of the array H. LDH >= max(1, N).

W Input parameter.

W is COMPLEX

The eigenvalue of H whose corresponding right or left eigenvector is to be computed.

V Input and output parameter.

V is COMPLEX

V is an array, dimension (N). On entry, if NOINIT = .FALSE., V must contain a starting vector for inverse iteration; otherwise V need not be set. On exit, V contains the computed eigenvector, normalized so that the component of largest magnitude has magnitude 1; here the magnitude of a complex number (x,y) is taken to be |x| + |y|.

B Output parameter.

B is COMPLEX

B is an array, dimension (LDB, N) .

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (N) .

EPS3 Input parameter.

EPS3 is REAL

A small machine-dependent value which is used to perturb close eigenvalues, and to replace zero pivots.

SMLNUM Input parameter.

SMLNUM is REAL

A machine-dependent value close to the underflow threshold.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit = 1: inverse iteration did not converge; V is set to the last iterate.

Related Information

For this routine in other precisions, please see *dlaein*, *slaein* and *zlaein*.

4.17.63 claesy

`claesy` computes the eigendecomposition of a 2-by-2 symmetric matrix

```
( ( A, B );( B, C ) )
```

provided the norm of the matrix of eigenvectors is larger than some threshold value.

RT1 is the eigenvalue of larger absolute value, and RT2 of smaller absolute value. If the eigenvectors are computed, then on return (CS1, SN1) is the unit eigenvector for RT1, hence

[CS1 SN1] . [A B] . [CS1 -SN1] = [RT1 0] [-SN1 CS1] [B C] [SN1 CS1] [0 RT2]

Syntax

Fortran specification:

```
use armpl_library

subroutine claesy(A, B, C, RT1, RT2, EVSCAL, CS1, SN1)
```

C specification:

```
#include "armpl.h"

void claesy_(const armpl_singlecomplex_t *a, const armpl_singlecomplex_t *b,
             const armpl_singlecomplex_t *c, armpl_singlecomplex_t *rt1,
             armpl_singlecomplex_t *rt2, armpl_singlecomplex_t *evscal,
             armpl_singlecomplex_t *cs1, armpl_singlecomplex_t *sn1);
```

Parameters

A Input parameter.

`A` is `COMPLEX`

The (1, 1) element of input matrix.

B Input parameter.

`B` is `COMPLEX`

The (1, 2) element of input matrix. The (2, 1) element is also given by `B`, since the 2-by-2 matrix is symmetric.

C Input parameter.

`C` is `COMPLEX`

The (2, 2) element of input matrix.

RT1 Output parameter.

`RT1` is `COMPLEX`

The eigenvalue of larger modulus.

RT2 Output parameter.

RT2 is COMPLEX

The eigenvalue of smaller modulus.

EVSCAL Output parameter.

EVSCAL is COMPLEX

The complex value by which the eigenvector matrix was scaled to make it orthonormal. If EVSCAL is zero, the eigenvectors were not computed. This means one of two things: the 2-by-2 matrix could not be diagonalized, or the norm of the matrix of eigenvectors before scaling was larger than the threshold value THRESH (set below).

CS1 Output parameter.

CS1 is COMPLEX

SN1 Output parameter.

SN1 is COMPLEX

If EVSCAL .NE. 0, (CS1, SN1) is the unit right eigenvector for RT1.

Related Information

For this routine in other precisions, please see *zlaesy*.

4.17.64 claev2

claev2 computes the eigendecomposition of a 2-by-2 Hermitian matrix

```
[  A          B  ]
[  CONJG(B)   C  ].
```

On return, RT1 is the eigenvalue of larger absolute value, RT2 is the eigenvalue of smaller absolute value, and (CS1,SN1) is the unit right eigenvector for RT1, giving the decomposition

[CS1 CONJG(SN1)] [A B] [CS1 -CONJG(SN1)] = [RT1 0] [-SN1 CS1] [CONJG(B) C] [SN1 CS1] [0 RT2].

Syntax

Fortran specification:

```
use armpl_library

subroutine claev2(A, B, C, RT1, RT2, CS1, SN1)
```

C specification:

```
#include "armpl.h"

void claev2_(const armpl_singlecomplex_t *a, const armpl_singlecomplex_t *b,
             const armpl_singlecomplex_t *c, float *rt1, float *rt2,
             float *cs1, armpl_singlecomplex_t *sn1);
```

Parameters

A Input parameter.

A is COMPLEX

The (1,1) element of the 2-by-2 matrix.

B Input parameter.

B is COMPLEX

The (1,2) element and the conjugate of the (2,1) element of the 2-by-2 matrix.

C Input parameter.

C is COMPLEX

The (2,2) element of the 2-by-2 matrix.

RT1 Output parameter.

RT1 is REAL

The eigenvalue of larger absolute value.

RT2 Output parameter.

RT2 is REAL

The eigenvalue of smaller absolute value.

CS1 Output parameter.

CS1 is REAL

SN1 Output parameter.

SN1 is COMPLEX

The vector (CS1, SN1) is a unit right eigenvector for RT1.

Related Information

For this routine in other precisions, please see *dlaev2*, *slaev2* and *zlaev2*.

4.17.65 clag2z

clag2z converts a COMPLEX matrix, SA, to a COMPLEX*16 matrix, A.

Note that while it is possible to overflow while converting from double to single, it is not possible to overflow when converting from single to double.

This is an auxiliary routine so there is no argument checking.

Syntax

Fortran specification:

```
use armpl_library

subroutine clag2z(M, N, SA, LDSA, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void clag2z_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_singlecomplex_t *sa, const armpl_int_t *ldsa,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of lines of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

SA Input parameter.

SA is COMPLEX

SA is an array, dimension (LDSA, N). On entry, the M-by-N coefficient matrix SA.

LDSA Input parameter.

LDSA is INTEGER

The leading dimension of the array SA. LDSA >= max(1, M).

A Output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On exit, the M-by-N coefficient matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit

Related Information

It also exists with a native C interface as *LAPACKE_clag2z*.

4.17.66 clags2

clags2 computes 2-by-2 unitary matrices U, V and Q, such that if (UPPER) then

```
U**H *A*Q = U**H *( A1 A2 )*Q = ( x  0  )
                  ( 0  A3 )     ( x  x  )
```

and

```
V**H*B*Q = V**H *( B1 B2 )*Q = ( x  0  )
                 ( 0  B3 )     ( x  x  )
```

or if (.NOT.UPPER) then

```
U**H *A*Q = U**H *( A1 0  )*Q = ( x  x  )
                  ( A2 A3 )     ( 0  x  )
```

and

```
V**H *B*Q = V**H *( B1 0  )*Q = ( x  x  )
                  ( B2 B3 )     ( 0  x  )
```

where

```
U = (   CSU     SNU ), V = (  CSV     SNV ),
    ( -SNU**H   CSU )      ( -SNV**H CSV )

Q = (   CSQ     SNQ )
    ( -SNQ**H   CSQ )
```

The rows of the transformed A and B are parallel. Moreover, if the input 2-by-2 matrix A is not zero, then the transformed (1,1) entry of A is not zero. If the input matrices A and B are both not zero, then the transformed (2,2) element of B is not zero, except when the first rows of input A and B are parallel and the second rows are zero.

Syntax

Fortran specification:

```
use armpl_library

subroutine clags2(UPPER, A1, A2, A3, B1, B2, B3, CSU, SNU, CSV, SNV, CSQ,
                  SNQ)
```

C specification:

```
#include "armpl.h"

void clags2_(const armpl_int_t *upper, const float *a1,
             const armpl_singlecomplex_t *a2, const float *a3,
             const float *b1, const armpl_singlecomplex_t *b2,
             const float *b3, float *csu, armpl_singlecomplex_t *snu,
             float *csv, armpl_singlecomplex_t *snv, float *csq,
             armpl_singlecomplex_t *snq);
```

Parameters

UPPER Input parameter.

UPPER is LOGICAL

= .TRUE.: the input matrices A and B are upper triangular. = .FALSE.: the input matrices A and B are lower triangular.

A1 Input parameter.

A1 is REAL

A2 Input parameter.

A2 is COMPLEX

A3 Input parameter.

A3 is REAL

On entry, A1, A2 and A3 are elements of the input 2-by-2 upper (lower) triangular matrix A.

B1 Input parameter.

B1 is REAL

B2 Input parameter.

B2 is COMPLEX

B3 Input parameter.

B3 is REAL

On entry, B1, B2 and B3 are elements of the input 2-by-2 upper (lower) triangular matrix B.

CSU Output parameter.

CSU is REAL

SNU Output parameter.

SNU is COMPLEX

The desired unitary matrix U.

CSV Output parameter.

CSV is REAL

SNV Output parameter.

SNV is COMPLEX

The desired unitary matrix V.

CSQ Output parameter.

CSQ is REAL

SNQ Output parameter.

SNQ is COMPLEX

The desired unitary matrix Q.

Related Information

For this routine in other precisions, please see *dlags2*, *slags2* and *zlags2*.

4.17.67 clagtm

clagtm performs a matrix-vector product of the form

```
B := alpha * A * X + beta * B
```

where A is a tridiagonal matrix of order N, B and X are N by NRHS matrices, and alpha and beta are real scalars, each of which may be 0., 1., or -1.

Syntax

Fortran specification:

```
use armpl_library

subroutine clagtm(TRANS, N, NRHS, ALPHA, DL, D, DU, X, LDX, BETA, B, LDB)
```

C specification:

```
#include "armpl.h"

void clagtm_(const char *trans, const armpl_int_t *n, const armpl_int_t *nrhs,
             const float *alpha, const armpl_singlecomplex_t *dl,
             const armpl_singlecomplex_t *d, const armpl_singlecomplex_t *du,
             const armpl_singlecomplex_t *x, const armpl_int_t *ldx,
             const float *beta, armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the operation applied to A. = 'N': No transpose, B := alpha * A * X + beta * B = 'T': Transpose, B := alpha * A^T * X + beta * B = 'C': Conjugate transpose, B := alpha * A^H * X + beta * B

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices X and B.

ALPHA Input parameter.

ALPHA is REAL

The scalar alpha. ALPHA must be 0., 1., or -1.; otherwise, it is assumed to be 0.

DL Input parameter.

DL is COMPLEX

DL is an array, dimension (N-1). The (n-1) sub-diagonal elements of T.

D Input parameter.

D is COMPLEX

D is an array, dimension (N). The diagonal elements of T.

DU Input parameter.

DU is COMPLEX

DU is an array, dimension (N-1). The (n-1) super-diagonal elements of T.

X Input parameter.

X is COMPLEX

X is an array, dimension (LDX, NRHS). The N by NRHS matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(N,1).

BETA Input parameter.

BETA is REAL

The scalar beta. BETA must be 0., 1., or -1.; otherwise, it is assumed to be 1.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the N by NRHS matrix B. On exit, B is overwritten by the matrix expression B := alpha * A * X + beta * B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(N,1).

Related Information

For this routine in other precisions, please see *dlagtm*, *slagtm* and *zlagtm*.

4.17.68 clahef

clahef computes a partial factorization of a complex Hermitian matrix A using the Bunch-Kaufman diagonal pivoting method. The partial factorization has the form:

A = (I U12) (A11 0) (I 0) if UPLO = 'U', or:

```
( 0  U22 ) (  0   D  ) ( U12**H U22**H )
```

A = (L11 0) (D 0) ($L11^H$ $L21^H$) if UPLO = 'L'

```
( L21  I ) (  0  A22 ) (  0      I     )
```

where the order of D is at most NB. The actual order is returned in the argument KB, and is either NB or NB-1, or N if N <= NB. Note that U^H denotes the conjugate transpose of U.

clahef is an auxiliary routine called by CHETRF. It uses blocked code (calling Level 3 BLAS) to update the submatrix A11 (if UPLO = 'U') or A22 (if UPLO = 'L').

Syntax

Fortran specification:

```
use armpl_library

subroutine clahef(UPLO, N, NB, KB, A, LDA, IPIV, W, LDW, INFO)
```

C specification:

```
#include "armpl.h"

void clahef_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nb,
             armpl_int_t *kb, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *ipiv,
             armpl_singlecomplex_t *w, const armpl_int_t *ldw,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the Hermitian matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NB Input parameter.

NB is INTEGER

The maximum number of columns of the matrix A that should be factored. NB should be at least 2 to allow for 2-by-2 pivot blocks.

KB Output parameter.

KB is INTEGER

The number of columns of A that were actually factored. KB is either NB-1 or NB, or N if N <= NB.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, A contains details of the partial factorization.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D.

If UPLO = 'U': Only the last KB elements of IPIV are set.

If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block.

If UPLO = 'L': Only the first KB elements of IPIV are set.

If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

W Output parameter.

W is COMPLEX

W is an array, dimension (LDW, NB) .

LDW Input parameter.

LDW is INTEGER

The leading dimension of the array W. LDW >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit > 0: if INFO = k, D(k,k) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular.

Related Information

For this routine in other precisions, please see *zlahef*.

4.17.69 clahef_aa

CLAHEF_AA factorizes a panel of a complex hermitian matrix A using the Aasen's algorithm. The panel consists of a set of NB rows of A when UPLO is U, or a set of NB columns when UPLO is L.

In order to factorize the panel, the Aasen's algorithm requires the last row, or column, of the previous panel. The first row, or column, of A is set to be the first row, or column, of an identity matrix, which is used to factorize the first panel.

The resulting J-th row of U, or J-th column of L, is stored in the (J-1)-th row, or column, of A (without the unit diagonals), while the diagonal and subdiagonal of A are overwritten by those of T.

Syntax

Fortran specification:

```
use armpl_library

subroutine clahef_aa(UPLO, J1, M, NB, A, LDA, IPIV, H, LDH, WORK)
```

C specification:

```
#include "armpl.h"

void clahef_aa_(const char *uplo, const armpl_int_t *j1, const armpl_int_t *m,
                const armpl_int_t *nb, armpl_singlecomplex_t *a,
                const armpl_int_t *lda, armpl_int_t *ipiv,
                armpl_singlecomplex_t *h, const armpl_int_t *ldh,
                armpl_singlecomplex_t *work, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

J1 Input parameter.

J1 is INTEGER

The location of the first row, or column, of the panel within the submatrix of A, passed to this routine, e.g., when called by CHETRF_AA, for the first panel, J1 is 1, while for the remaining panels, J1 is 2.

M Input parameter.

M is INTEGER

The dimension of the submatrix. M >= 0.

NB Input parameter.

NB is INTEGER

The dimension of the panel to be facotorized.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, M) for. the first panel, while dimension (LDA,M+1) for the remaining panels.

On entry, A contains the last row, or column, of the previous panel, and the trailing submatrix of A to be factorized, except for the first panel, only the panel is passed.

On exit, the leading panel is factorized.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1,N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the row and column interchanges, the row and column k were interchanged with the row and column IPIV(k).

H Input and output parameter.

H is COMPLEX workspace, dimension (LDH, NB).

LDH Input parameter.

LDH is INTEGER

The leading dimension of the workspace H. LDH >= max(1, M).

WORK Output parameter.

WORK is COMPLEX workspace, dimension (M).

Related Information

For this routine in other precisions, please see *zlahef_aa*.

4.17.70 clahef_rook

CLAHEF_ROOK computes a partial factorization of a complex Hermitian matrix A using the bounded Bunch-Kaufman ("rook") diagonal pivoting method. The partial factorization has the form:

A = (I U12) (A11 0) (I 0) if UPLO = 'U', or:

```
( 0  U22 ) (  0   D  ) ( U12**H U22**H )
```

A = (L11 0) (D 0) ($L11^H$ $L21^H$) if UPLO = 'L'

```
( L21  I ) (  0  A22 ) (  0      I     )
```

where the order of D is at most NB. The actual order is returned in the argument KB, and is either NB or NB-1, or N if N <= NB. Note that U^H denotes the conjugate transpose of U.

CLAHEF_ROOK is an auxiliary routine called by CHETRF_ROOK. It uses blocked code (calling Level 3 BLAS) to update the submatrix A11 (if UPLO = 'U') or A22 (if UPLO = 'L').

Syntax

Fortran specification:

```
use armpl_library

subroutine clahef_rook(UPLO, N, NB, KB, A, LDA, IPIV, W, LDW, INFO)
```

C specification:

```
#include "armpl.h"

void clahef_rook_(const char *uplo, const armpl_int_t *n,
                  const armpl_int_t *nb, armpl_int_t *kb,
                  armpl_singlecomplex_t *a, const armpl_int_t *lda,
                  armpl_int_t *ipiv, armpl_singlecomplex_t *w,
                  const armpl_int_t *ldw, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the Hermitian matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NB Input parameter.

NB is INTEGER

The maximum number of columns of the matrix A that should be factored. NB should be at least 2 to allow for 2-by-2 pivot blocks.

KB Output parameter.

KB is INTEGER

The number of columns of A that were actually factored. KB is either NB-1 or NB, or N if N <= NB.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, A contains details of the partial factorization.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D.

If UPLO = 'U': Only the last KB elements of IPIV are set.

If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k-1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k-1 and -IPIV(k-1) were inerchaged, D(k-1:k,k-1:k) is a 2-by-2 diagonal block.

If UPLO = 'L': Only the first KB elements of IPIV are set.

If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k+1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k+1 and -IPIV(k+1) were inerchaged, D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

W Output parameter.

W is COMPLEX

W is an array, dimension (LDW, NB) .

LDW Input parameter.

LDW is INTEGER

The leading dimension of the array W. LDW >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit > 0: if INFO = k, D(k,k) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular.

Related Information

For this routine in other precisions, please see *zlahef_rook*.

4.17.71 clahqr

```
CLAHQR is an auxiliary routine called by CHSEQR to update the
   eigenvalues and Schur decomposition already computed by CHSEQR, by
   dealing with the Hessenberg submatrix in rows and columns ILO to
   IHI.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine clahqr(WANTT, WANTZ, N, ILO, IHI, H, LDH, W, ILOZ, IHIZ, Z, LDZ,
                  INFO)
```

C specification:

```
#include "armpl.h"

void clahqr_(const armpl_int_t *wantt, const armpl_int_t *wantz,
             const armpl_int_t *n, const armpl_int_t *ilo,
             const armpl_int_t *ihi, armpl_singlecomplex_t *h,
             const armpl_int_t *ldh, armpl_singlecomplex_t *w,
             const armpl_int_t *iloz, const armpl_int_t *ihiz,
```

(continues on next page)

(continued from previous page)

```
            armpl_singlecomplex_t *z, const armpl_int_t *ldz,
            armpl_int_t *info);
```

Parameters

WANTT Input parameter.

WANTT is LOGICAL

= .TRUE. : the full Schur form T is required; = .FALSE.: only eigenvalues are required.

WANTZ Input parameter.

WANTZ is LOGICAL

= .TRUE. : the matrix of Schur vectors Z is required; = .FALSE.: Schur vectors are not required.

N Input parameter.

N is INTEGER

The order of the matrix H. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

It is assumed that H is already upper triangular in rows and columns IHI+1:N, and that H(ILO,ILO-1) = 0 (unless ILO = 1). CLAHQR works primarily with the Hessenberg submatrix in rows and columns ILO to IHI, but applies transformations to all of H if WANTT is .TRUE.. 1 <= ILO <= max(1, IHI); IHI <= N.

H Input and output parameter.

H is COMPLEX

H is an array, dimension (LDH, N). On entry, the upper Hessenberg matrix H. On exit, if INFO is zero and if WANTT is .TRUE., then H is upper triangular in rows and columns ILO:IHI. If INFO is zero and if WANTT is .FALSE., then the contents of H are unspecified on exit. The output state of H in case INF is positive is below under the description of INFO.

LDH Input parameter.

LDH is INTEGER

The leading dimension of the array H. LDH >= max(1, N).

W Output parameter.

W is COMPLEX

W is an array, dimension (N). The computed eigenvalues ILO to IHI are stored in the corresponding elements of W. If WANTT is .TRUE., the eigenvalues are stored in the same order as on the diagonal of the Schur form returned in H, with W(i) = H(i,i).

ILOZ Input parameter.

ILOZ is INTEGER

IHIZ Input parameter.

IHIZ is INTEGER

Specify the rows of Z to which transformations must be applied if WANTZ is .TRUE.. 1 <= ILOZ <= ILO; IHI <= IHIZ <= N.

Z Input and output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, N). If WANTZ is .TRUE., on entry Z must contain the current matrix Z of transformations accumulated by CHSEQR, and on exit Z has been updated; transformations are applied only to the submatrix Z(ILOZ:IHIZ,ILO:IHI). If WANTZ is .FALSE., Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit .GT. 0: if INFO = i, CLAHQR failed to compute all the eigenvalues ILO to IHI in a total of 30 iterations per eigenvalue; elements i+1:ihi of W contain those eigenvalues which have been successfully computed.

If INFO .GT. 0 and WANTT is .FALSE., then on exit, the remaining unconverged eigenvalues are the eigenvalues of the upper Hessenberg matrix rows and columns ILO thorugh INFO of the final, output value of H.

If INFO .GT. 0 and WANTT is .TRUE., then on exit (*) (initial value of H)*U = U*(final value of H) where U is an orthognal matrix. The final value of H is upper Hessenberg and triangular in rows and columns INFO+1 through IHI.

If INFO .GT. 0 and WANTZ is .TRUE., then on exit (final value of Z) = (initial value of Z)*U where U is the orthogonal matrix in (*) (regardless of the value of WANTT.)

Related Information

For this routine in other precisions, please see *dlahqr*, *slahqr* and *zlahqr*.

4.17.72 clahr2

clahr2 reduces the first NB columns of A complex general n-BY-(n-k+1) matrix A so that elements below the k-th subdiagonal are zero. The reduction is performed by an unitary similarity transformation Q^H * A * Q. The routine returns the matrices V and T which determine Q as a block reflector I - V*T*v^H , and also the matrix Y = A * V * T.

This is an auxiliary routine called by CGEHRD.

Syntax

Fortran specification:

```
use armpl_library

subroutine clahr2(N, K, NB, A, LDA, TAU, T, LDT, Y, LDY)
```

C specification:

```
#include "armpl.h"

void clahr2_(const armpl_int_t *n, const armpl_int_t *k,
             const armpl_int_t *nb, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_singlecomplex_t *tau,
             armpl_singlecomplex_t *t, const armpl_int_t *ldt,
             armpl_singlecomplex_t *y, const armpl_int_t *ldy);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A.

K Input parameter.

K is INTEGER

The offset for the reduction. Elements below the k-th subdiagonal in the first NB columns are reduced to zero. K < N.

NB Input parameter.

NB is INTEGER

The number of columns to be reduced.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA,N-K+1). On entry, the n-by-(n-k+1) general matrix A. On exit, the elements on and above the k-th subdiagonal in the first NB columns are overwritten with the corresponding elements of the reduced matrix; the elements below the k-th subdiagonal, with the array TAU, represent the matrix Q as a product of elementary reflectors. The other columns of A are unchanged. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TAU Output parameter.

TAU is COMPLEX

TAU is an array, dimension (NB). The scalar factors of the elementary reflectors. See Further Details.

T Output parameter.

T is COMPLEX

T is an array, dimension (LDT, NB). The upper triangular matrix T.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= NB.

Y Output parameter.

Y is COMPLEX

Y is an array, dimension (LDY, NB). The n-by-nb matrix Y.

LDY Input parameter.

LDY is INTEGER

The leading dimension of the array Y. LDY >= N.

Related Information

For this routine in other precisions, please see *dlahr2*, *slahr2* and *zlahr2*.

4.17.73 claic1

`claic1` applies one step of incremental condition estimation in its simplest version:

Let x, twonorm(x) = 1, be an approximate singular vector of an j-by-j lower triangular matrix L, such that

```
twonorm(L*x) = sest
```

Then `claic1` computes sestpr, s, c such that the vector

```
       [ s*x ]
xhat = [  c  ]
```

is an approximate singular vector of

```
       [ L       0    ]
Lhat = [ w**H gamma ]
```

in the sense that

```
twonorm(Lhat*xhat) = sestpr.
```

Depending on JOB, an estimate for the largest or smallest singular value is computed.

Note that [s c]^H and sestpr**2 is an eigenpair of the system

```
diag(sest*sest, 0) + [alpha  gamma] * [ conjg(alpha) ]
                                      [ conjg(gamma) ]
```

where alpha = x^H *w.

Syntax

Fortran specification:

```
use armpl_library

subroutine claic1(JOB, J, X, SEST, W, GAMMA, SESTPR, S, C)
```

C specification:

```
#include "armpl.h"

void claic1_(const armpl_int_t *job, const armpl_int_t *j,
             const armpl_singlecomplex_t *x, const float *sest,
             const armpl_singlecomplex_t *w,
             const armpl_singlecomplex_t *gamma, float *sestpr,
             armpl_singlecomplex_t *s, armpl_singlecomplex_t *c);
```

Parameters

JOB Input parameter.

`JOB` is `INTEGER`

= 1: an estimate for the largest singular value is computed. = 2: an estimate for the smallest singular value is computed.

J Input parameter.

`J` is `INTEGER`

Length of `X` and `W`

X Input parameter.

X is COMPLEX

X is an array, dimension (J). The j-vector x.

SEST Input parameter.

SEST is REAL

Estimated singular value of j by j matrix L

W Input parameter.

W is COMPLEX

W is an array, dimension (J). The j-vector w.

GAMMA Input parameter.

GAMMA is COMPLEX

The diagonal element gamma.

SESTPR Output parameter.

SESTPR is REAL

Estimated singular value of (j+1) by (j+1) matrix Lhat.

S Output parameter.

S is COMPLEX

Sine needed in forming xhat.

C Output parameter.

C is COMPLEX

Cosine needed in forming xhat.

Related Information

For this routine in other precisions, please see *dlaic1*, *slaic1* and *zlaic1*.

4.17.74 clals0

clals0 applies back the multiplying factors of either the left or the right singular vector matrix of a diagonal matrix appended by a row to the right hand side matrix B in solving the least squares problem using the divide-and-conquer SVD approach.

For the left singular vector matrix, three types of orthogonal matrices are involved:

(1L) Givens rotations: the number of such rotations is GIVPTR; the

```
pairs of columns/rows they were applied to are stored in GIVCOL;
and the C- and S-values of these rotations are stored in GIVNUM.
```

(2L) Permutation. The (NL+1)-st row of B is to be moved to the first

```
row, and for J=2:N, PERM(J)-th row of B is to be moved to the
J-th row.
```

(3L) The left singular vector matrix of the remaining matrix.

For the right singular vector matrix, four types of orthogonal matrices are involved:

(1R) The right singular vector matrix of the remaining matrix.

(2R) If SQRE = 1, one extra Givens rotation to generate the right

```
null space.
```

(3R) The inverse transformation of (2L).

(4R) The inverse transformation of (1L).

Syntax

Fortran specification:

```
use armpl_library

subroutine clals0(ICOMPQ, NL, NR, SQRE, NRHS, B, LDB, BX, LDBX, PERM, GIVPTR,
                  GIVCOL, LDGCOL, GIVNUM, LDGNUM, POLES, DIFL, DIFR, Z, K, C,
                  S, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void clals0_(const armpl_int_t *icompq, const armpl_int_t *nl,
             const armpl_int_t *nr, const armpl_int_t *sqre,
             const armpl_int_t *nrhs, armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, armpl_singlecomplex_t *bx,
             const armpl_int_t *ldbx, const armpl_int_t *perm,
             const armpl_int_t *givptr, const armpl_int_t *givcol,
             const armpl_int_t *ldgcol, const float *givnum,
             const armpl_int_t *ldgnum, const float *poles, const float *difl,
             const float *difr, const float *z, const armpl_int_t *k,
             const float *c, const float *s, float *rwork,
             armpl_int_t *info);
```

Parameters

ICOMPQ Input parameter.

ICOMPQ is INTEGER

Specifies whether singular vectors are to be computed in factored form: = 0: Left singular vector matrix. = 1: Right singular vector matrix.

NL Input parameter.

NL is INTEGER

The row dimension of the upper block. NL >= 1.

NR Input parameter.

NR is INTEGER

The row dimension of the lower block. NR >= 1.

SQRE Input parameter.

SQRE is INTEGER

= 0: the lower block is an NR-by-NR square matrix. = 1: the lower block is an NR-by-(NR+1) rectangular matrix.

The bidiagonal matrix has row dimension N = NL + NR + 1, and column dimension M = N + SQRE.

NRHS Input parameter.

NRHS is INTEGER

The number of columns of B and BX. NRHS must be at least 1.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On input, B contains the right hand sides of the least squares problem in rows 1 through M. On output, B contains the solution X in rows 1 through N.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B. LDB must be at least max(1,MAX(M, N)).

BX Output parameter.

BX is COMPLEX

BX is an array, dimension (LDBX, NRHS) .

LDBX Input parameter.

LDBX is INTEGER

The leading dimension of BX.

PERM Input parameter.

PERM is INTEGER array, dimension (N)

The permutations (from deflation and sorting) applied to the two blocks.

GIVPTR Input parameter.

GIVPTR is INTEGER

The number of Givens rotations which took place in this subproblem.

GIVCOL Input parameter.

GIVCOL is INTEGER array, dimension (LDGCOL, 2)

Each pair of numbers indicates a pair of rows/columns involved in a Givens rotation.

LDGCOL Input parameter.

LDGCOL is INTEGER

The leading dimension of GIVCOL, must be at least N.

GIVNUM Input parameter.

GIVNUM is REAL

GIVNUM is an array, dimension (LDGNUM, 2). Each number indicates the C or S value used in the corresponding Givens rotation.

LDGNUM Input parameter.

LDGNUM is INTEGER

The leading dimension of arrays DIFR, POLES and GIVNUM, must be at least K.

POLES Input parameter.

POLES is REAL

POLES is an array, dimension (LDGNUM, 2). On entry, POLES(1:K, 1) contains the new singular values obtained from solving the secular equation, and POLES(1:K, 2) is an array containing the poles in the secular equation.

DIFL Input parameter.

DIFL is REAL

DIFL is an array, dimension (K).. On entry, DIFL(I) is the distance between I-th updated (undeflated) singular value and the I-th (undeflated) old singular value.

DIFR Input parameter.

DIFR is REAL

DIFR is an array, dimension (LDGNUM, 2).. On entry, DIFR(I, 1) contains the distances between I-th updated (undeflated) singular value and the I+1-th (undeflated) old singular value. And DIFR(I, 2) is the normalizing factor for the I-th right singular vector.

Z Input parameter.

Z is REAL

Z is an array, dimension (K). Contain the components of the deflation-adjusted updating row vector.

K Input parameter.

K is INTEGER

Contains the dimension of the non-deflated matrix, This is the order of the related secular equation. 1 <= K <=N.

C Input parameter.

C is REAL

C contains garbage if SQRE =0 and the C-value of a Givens rotation related to the right null space if SQRE = 1.

S Input parameter.

S is REAL

S contains garbage if SQRE =0 and the S-value of a Givens rotation related to the right null space if SQRE = 1.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension. (K*(1+NRHS) + 2* NRHS)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *dlals0*, *slals0* and *zlals0*.

4.17.75 clalsa

clalsa is an itermediate step in solving the least squares problem by computing the SVD of the coefficient matrix in compact form (The singular vectors are computed as products of simple orthorgonal matrices.).

If ICOMPQ = 0, clalsa applies the inverse of the left singular vector matrix of an upper bidiagonal matrix to the right hand side; and if ICOMPQ = 1, clalsa applies the right singular vector matrix to the right hand side. The singular vector matrices were generated in compact form by clalsa.

Syntax

Fortran specification:

```
use armpl_library

subroutine clalsa(ICOMPQ, SMLSIZ, N, NRHS, B, LDB, BX, LDBX, U, LDU, VT, K,
                  DIFL, DIFR, Z, POLES, GIVPTR, GIVCOL, LDGCOL, PERM, GIVNUM,
                  C, S, RWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void clalsa_(const armpl_int_t *icompq, const armpl_int_t *smlsiz,
             const armpl_int_t *n, const armpl_int_t *nrhs,
             armpl_singlecomplex_t *b, const armpl_int_t *ldb,
             armpl_singlecomplex_t *bx, const armpl_int_t *ldbx,
             const float *u, const armpl_int_t *ldu, const float *vt,
             const armpl_int_t *k, const float *difl, const float *difr,
             const float *z, const float *poles, const armpl_int_t *givptr,
             const armpl_int_t *givcol, const armpl_int_t *ldgcol,
             const armpl_int_t *perm, const float *givnum, const float *c,
             const float *s, float *rwork, armpl_int_t *iwork,
             armpl_int_t *info);
```

Parameters

ICOMPQ Input parameter.

ICOMPQ is INTEGER

Specifies whether the left or the right singular vector matrix is involved. = 0: Left singular vector matrix = 1: Right singular vector matrix

SMLSIZ Input parameter.

SMLSIZ is INTEGER

The maximum size of the subproblems at the bottom of the computation tree.

N Input parameter.

N is INTEGER

The row and column dimensions of the upper bidiagonal matrix.

NRHS Input parameter.

NRHS is INTEGER

The number of columns of B and BX. NRHS must be at least 1.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On input, B contains the right hand sides of the least squares problem in rows 1 through M. On output, B contains the solution X in rows 1 through N.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B in the calling subprogram. LDB must be at least max(1,MAX(M, N)).

BX Output parameter.

BX is COMPLEX

BX is an array, dimension (LDBX, NRHS). On exit, the result of applying the left or right singular vector matrix to B.

LDBX Input parameter.

LDBX is INTEGER

The leading dimension of BX.

U Input parameter.

U is REAL

U is an array, dimension (LDU, SMLSIZ).. On entry, U contains the left singular vector matrices of all subproblems at the bottom level.

LDU Input parameter.

LDU is INTEGER, LDU = > N.

The leading dimension of arrays U, VT, DIFL, DIFR, POLES, GIVNUM, and Z.

VT Input parameter.

VT is REAL

VT is an array, dimension (LDU, SMLSIZ+1).. On entry, VT^H contains the right singular vector matrices of all subproblems at the bottom level.

K Input parameter.

K is INTEGER array, dimension (N).

DIFL Input parameter.

DIFL is REAL

DIFL is an array, dimension (LDU, NLVL).. where NLVL = INT(log_2 (N/(SMLSIZ+1))) + 1.

DIFR Input parameter.

DIFR is REAL

DIFR is an array, dimension (LDU, 2 * NLVL).. On entry, DIFL(*, I) and DIFR(*, 2 * I -1) record distances between singular values on the I-th level and singular values on the (I -1)-th level, and DIFR(*, 2 * I) record the normalizing factors of the right singular vectors matrices of subproblems on I-th level.

Z Input parameter.

Z is REAL

Z is an array, dimension (LDU, NLVL).. On entry, Z(1, I) contains the components of the deflation- adjusted updating row vector for subproblems on the I-th level.

POLES Input parameter.

POLES is REAL

POLES is an array, dimension (LDU, 2 * NLVL).. On entry, POLES(*, 2 * I -1: 2 * I) contains the new and old singular values involved in the secular equations on the I-th level.

GIVPTR Input parameter.

GIVPTR is INTEGER array, dimension (N).

On entry, GIVPTR(I) records the number of Givens rotations performed on the I-th problem on the computation tree.

GIVCOL Input parameter.

GIVCOL is INTEGER array, dimension (LDGCOL, 2 * NLVL).

On entry, for each I, GIVCOL(*, 2 * I - 1: 2 * I) records the locations of Givens rotations performed on the I-th level on the computation tree.

LDGCOL Input parameter.

LDGCOL is INTEGER, LDGCOL = > N.

The leading dimension of arrays GIVCOL and PERM.

PERM Input parameter.

PERM is INTEGER array, dimension (LDGCOL, NLVL).

On entry, PERM(*, I) records permutations done on the I-th level of the computation tree.

GIVNUM Input parameter.

GIVNUM is REAL

GIVNUM is an array, dimension (LDU, 2 * NLVL).. On entry, GIVNUM(*, 2 *I -1 : 2 * I) records the C- and S- values of Givens rotations performed on the I-th level on the computation tree.

C Input parameter.

C is REAL

C is an array, dimension (N).. On entry, if the I-th subproblem is not square, C(I) contains the C-value of a Givens rotation related to the right null space of the I-th subproblem.

S Input parameter.

S is REAL

S is an array, dimension (N).. On entry, if the I-th subproblem is not square, S(I) contains the S-value of a Givens rotation related to the right null space of the I-th subproblem.

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension at least. MAX((SMLSZ+1)*NRHS*3, N*(1+NRHS) + 2* NRHS).

IWORK Output parameter.

IWORK is INTEGER array, dimension (3*N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *dlalsa*, *slalsa* and *zlalsa*.

4.17.76 clalsd

clalsd uses the singular value decomposition of A to solve the least squares problem of finding X to minimize the Euclidean norm of each column of A*X-B, where A is N-by-N upper bidiagonal, and X and B are N-by-NRHS. The solution X overwrites B.

The singular values of A smaller than RCOND times the largest singular value are treated as zero in solving the least squares problem; in this case a minimum norm solution is returned. The actual singular values are returned in D in ascending order.

This code makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray XMP, Cray YMP, Cray C 90, or Cray 2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine clalsd(UPLO, SMLSIZ, N, NRHS, D, E, B, LDB, RCOND, RANK, WORK,
                  RWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void clalsd_(const char *uplo, const armpl_int_t *smlsiz,
             const armpl_int_t *n, const armpl_int_t *nrhs, float *d,
             float *e, armpl_singlecomplex_t *b, const armpl_int_t *ldb,
             const float *rcond, armpl_int_t *rank,
             armpl_singlecomplex_t *work, float *rwork, armpl_int_t *iwork,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': D and E define an upper bidiagonal matrix. = 'L': D and E define a lower bidiagonal matrix.

SMLSIZ Input parameter.

SMLSIZ is INTEGER

The maximum size of the subproblems at the bottom of the computation tree.

N Input parameter.

N is INTEGER

The dimension of the bidiagonal matrix. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of columns of B. NRHS must be at least 1.

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry D contains the main diagonal of the bidiagonal matrix. On exit, if INFO = 0, D contains its singular values.

E Input and output parameter.

E is REAL

E is an array, dimension (N-1). Contains the super-diagonal entries of the bidiagonal matrix. On exit, E has been destroyed.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On input, B contains the right hand sides of the least squares problem. On output, B contains the solution X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B in the calling subprogram. LDB must be at least max(1, N).

RCOND Input parameter.

RCOND is REAL

The singular values of A less than or equal to RCOND times the largest singular value are treated as zero in solving the least squares problem. If RCOND is negative, machine precision is used instead. For example, if diag(S)*X=B were the least squares problem, where diag(S) is a diagonal matrix of singular values, the solution would be X(i) = B(i) / S(i) if S(i) is greater than RCOND*max(S), and X(i) = 0 if S(i) is less than or equal to RCOND*max(S).

RANK Output parameter.

RANK is INTEGER

The number of singular values of A greater than RCOND times the largest singular value.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (N * NRHS). .

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension at least. (9* N + 2*N* SMLSIZ + 8*N*NLVL + 3*SMLSIZ* NRHS + MAX((SMLSIZ+1)**2, N*(1+NRHS) + 2* NRHS), where NLVL = MAX(0, INT(LOG_2(MIN(M,N)/(SMLSIZ+1))) + 1)

IWORK Output parameter.

IWORK is INTEGER array, dimension (3*N*NLVL + 11*N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: The algorithm failed to compute a singular value while working on the submatrix lying in rows and columns INFO/(N+1) through MOD(INFO,N+1).

Related Information

For this routine in other precisions, please see *dlalsd*, *slalsd* and *zlalsd*.

4.17.77 clamswlq

CLAMQRTS overwrites the general real M-by-N matrix C with SIDE = 'L' SIDE = 'R' TRANS = 'N': Q * C C * Q TRANS = 'T': Q^H * C C * Q^H where Q is a real orthogonal matrix defined as the product of blocked elementary reflectors computed by short wide LQ factorization (CLASWLQ)

Syntax

Fortran specification:

```
use armpl_library

subroutine clamswlq(SIDE, TRANS, M, N, K, MB, NB, A, LDA, T, LDT, C, LDC,
                    WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void clamswlq_(const char *side, const char *trans, const armpl_int_t *m,
               const armpl_int_t *n, const armpl_int_t *k,
               const armpl_int_t *mb, const armpl_int_t *nb,
               armpl_singlecomplex_t *a, const armpl_int_t *lda,
               const armpl_singlecomplex_t *t, const armpl_int_t *ldt,
               armpl_singlecomplex_t *c, const armpl_int_t *ldc,
               armpl_singlecomplex_t *work, const armpl_int_t *lwork,
               armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left; = 'R': apply Q or Q^H from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'C': Transpose, apply Q^H .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >=0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= M.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. M >= K >= 0;

MB Input parameter.

MB is INTEGER

The row block size to be used in the blocked QR. M >= MB >= 1

NB Input parameter.

NB is INTEGER

The column block size to be used in the blocked QR. NB > M.

NB Input parameter.

NB is INTEGER

The block size to be used in the blocked QR. MB > M.

A Input parameter.

A is COMPLEX

A is an array, dimension. (LDA, M) if SIDE = 'L', (LDA, N) if SIDE = 'R' The i-th row must contain the vector which defines the blocked elementary reflector H(i), for i = 1,2,...,k, as returned by CLASWLQ in the first k rows of its array argument A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDA >= max(1, M); if SIDE = 'R', LDA >= max(1, N).

T Input parameter.

T is COMPLEX

T is an array, dimension. (M * Number of blocks(CEIL(N-K/NB-K)), The blocked upper triangular block reflectors stored in compact form as a sequence of upper triangular blocks. See below for further details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= MB.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^H * C or C*Q^H or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

(workspace) COMPLEX

(workspace) is an array, dimension (MAX(1, LWORK)) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, NB) * MB; if SIDE = 'R', LWORK >= max(1, M) * MB. If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dlamswlq*, *slamswlq* and *zlamswlq*.

4.17.78 clamtsqr

CLAMTSQR overwrites the general complex M-by-N matrix C with SIDE = 'L' SIDE = 'R' TRANS = 'N': Q * C C * Q TRANS = 'C': Q^H * C C * Q^H where Q is a real orthogonal matrix defined as the product of blocked elementary reflectors computed by tall skinny QR factorization (CLATSQR)

Syntax

Fortran specification:

```
use armpl_library

subroutine clamtsqr(SIDE, TRANS, M, N, K, MB, NB, A, LDA, T, LDT, C, LDC,
                    WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void clamtsqr_(const char *side, const char *trans, const armpl_int_t *m,
               const armpl_int_t *n, const armpl_int_t *k,
               const armpl_int_t *mb, const armpl_int_t *nb,
               armpl_singlecomplex_t *a, const armpl_int_t *lda,
               const armpl_singlecomplex_t *t, const armpl_int_t *ldt,
               armpl_singlecomplex_t *c, const armpl_int_t *ldc,
               armpl_singlecomplex_t *work, const armpl_int_t *lwork,
               armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left; = 'R': apply Q or Q^H from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'C': Conjugate Transpose, apply Q^H .

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >=0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. M >= N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. N >= K >= 0;

MB Input parameter.

MB is INTEGER

The block size to be used in the blocked QR. MB > N. (must be the same as DLATSQR)

NB Input parameter.

NB is INTEGER

The column block size to be used in the blocked QR. N >= NB >= 1.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, K). The i-th column must contain the vector which defines the blockedelementary reflector H(i), for i = 1,2,...,k, as returned by DLATSQR in the first k columns of its array argument A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDA >= max(1, M); if SIDE = 'R', LDA >= max(1, N).

T Input parameter.

T is COMPLEX

T is an array, dimension. (N * Number of blocks(CEIL(M-K/MB-K)), The blocked upper triangular block reflectors stored in compact form as a sequence of upper triangular blocks. See below for further details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= NB.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^H * C or C*Q^H or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

(workspace) COMPLEX

(workspace) is an array, dimension (MAX(1, LWORK)) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK.

If SIDE = 'L', LWORK >= max(1, N)*NB; if SIDE = 'R', LWORK >= max(1, MB)*NB. If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dlamtsqr*, *slamtsqr* and *zlamtsqr*.

4.17.79 clangb

clangb returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of an n by n band matrix A, with kl sub-diagonals and ku super-diagonals.

Syntax

Fortran specification:

```
use armpl_library

real function clangb(NORM, N, KL, KU, AB, LDAB, WORK)
```

C specification:

```
#include "armpl.h"

float clangb_(const char *norm, const armpl_int_t *n, const armpl_int_t *kl,
              const armpl_int_t *ku, const armpl_singlecomplex_t *ab,
              const armpl_int_t *ldab, float *work, ... );
```

Returns

CLANGB = (max(abs(A(i,j))), NORM = 'M' or 'm'

((norm1(A), NORM = '1', 'O' or 'o' ((normI(A), NORM = 'I' or 'i' ((normF(A), NORM = 'F', 'f', 'E' or 'e'

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in CLANGB as described above.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0. When N = 0, CLANGB is set to zero.

KL Input parameter.

KL is INTEGER

The number of sub-diagonals of the matrix A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of super-diagonals of the matrix A. KU >= 0.

AB Input parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). The band matrix A, stored in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(ku+1+i-j,j) = A(i,j) for max(1,j-ku)<=i<=min(n,j+kl).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KL+KU+1.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= N when NORM = 'I'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *dlangb*, *slangb* and *zlangb*.

4.17.80 clange

clange returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a complex matrix A.

Syntax

Fortran specification:

```
use armpl_library

real function clange(NORM, M, N, A, LDA, WORK)
```

C specification:

```
#include "armpl.h"

float clange_(const char *norm, const armpl_int_t *m, const armpl_int_t *n,
              const armpl_singlecomplex_t *a, const armpl_int_t *lda,
              float *work, ... );
```

Returns

CLANGE = (max(abs(A(i,j))), NORM = 'M' or 'm'

((norm1(A), NORM = '1', 'O' or 'o' ((normI(A), NORM = 'I' or 'i' ((normF(A), NORM = 'F', 'f', 'E' or 'e'

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in CLANGE as described above.

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0. When M = 0, CLANGE is set to zero.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0. When N = 0, CLANGE is set to zero.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The m by n matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(M,1).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= M when NORM = 'I'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *dlange*, *slange* and *zlange*. It also exists with a native C interface as *LAPACKE_clange*.

4.17.81 clangt

clangt returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a complex tridiagonal matrix A.

Syntax

Fortran specification:

```
use armpl_library

real function clangt(NORM, N, DL, D, DU)
```

C specification:

```
#include "armpl.h"

float clangt_(const char *norm, const armpl_int_t *n,
              const armpl_singlecomplex_t *dl, const armpl_singlecomplex_t *d,
              const armpl_singlecomplex_t *du, ... );
```

Returns

CLANGT = (max(abs(A(i,j))), NORM = 'M' or 'm'

((norm1(A), NORM = '1', 'O' or 'o' ((normI(A), NORM = 'I' or 'i' ((normF(A), NORM = 'F', 'f', 'E' or 'e'

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in CLANGT as described above.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0. When N = 0, CLANGT is set to zero.

DL Input parameter.

DL is COMPLEX

DL is an array, dimension (N-1). The (n-1) sub-diagonal elements of A.

D Input parameter.

D is COMPLEX

D is an array, dimension (N). The diagonal elements of A.

DU Input parameter.

DU is COMPLEX

DU is an array, dimension (N-1). The (n-1) super-diagonal elements of A.

Related Information

For this routine in other precisions, please see *dlangt*, *slangt* and *zlangt*.

4.17.82 clanhb

clanhb returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of an n by n hermitian band matrix A, with k super-diagonals.

Syntax

Fortran specification:

```
use armpl_library

real function clanhb(NORM, UPLO, N, K, AB, LDAB, WORK)
```

C specification:

```
#include "armpl.h"

float clanhb_(const char *norm, const char *uplo, const armpl_int_t *n,
              const armpl_int_t *k, const armpl_singlecomplex_t *ab,
              const armpl_int_t *ldab, float *work, ... );
```

Returns

CLANHB = (max(abs(A(i,j))), NORM = ‘M’ or ‘m’

((norm1(A), NORM = ‘1’, ‘O’ or ‘o’ ((normI(A), NORM = ‘I’ or ‘i’ ((normF(A), NORM = ‘F’, ‘f’, ‘E’ or ‘e’

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in CLANHB as described above.

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the band matrix A is supplied. = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0. When N = 0, CLANHB is set to zero.

K Input parameter.

K is INTEGER

The number of super-diagonals or sub-diagonals of the band matrix A. K >= 0.

AB Input parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). The upper or lower triangle of the hermitian band matrix A, stored in the first K+1 rows of AB. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(k+1+i-j,j) = A(i,j) for max(1,j-k)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+k). Note that the imaginary parts of the diagonal elements need not be set and are assumed to be zero.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= K+1.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= N when NORM = 'I' or '1' or 'O'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *zlanhb*.

4.17.83 clanhe

clanhe returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a complex hermitian matrix A.

Syntax

Fortran specification:

```
use armpl_library

real function clanhe(NORM, UPLO, N, A, LDA, WORK)
```

C specification:

```
#include "armpl.h"

float clanhe_(const char *norm, const char *uplo, const armpl_int_t *n,
              const armpl_singlecomplex_t *a, const armpl_int_t *lda,
              float *work, ... );
```

Returns

CLANHE = (max(abs(A(i,j))), NORM = ‘M’ or ‘m’

((norm1(A), NORM = ‘1’, ‘O’ or ‘o’ ((normI(A), NORM = ‘I’ or ‘i’ ((normF(A), NORM = ‘F’, ‘f’, ‘E’ or ‘e’

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in CLANHE as described above.

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the hermitian matrix A is to be referenced. = ‘U’: Upper triangular part of A is referenced = ‘L’: Lower triangular part of A is referenced

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0. When N = 0, CLANHE is set to zero.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The hermitian matrix A. If UPLO = ‘U’, the leading n by n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = ‘L’, the leading n by n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. Note that the imaginary parts of the diagonal elements need not be set and are assumed to be zero.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(N,1).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= N when NORM = 'I' or '1' or 'O'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *zlanhe*. It also exists with a native C interface as *LAPACKE_clanhe*.

4.17.84 clanhf

clanhf returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a complex Hermitian matrix A in RFP format.

Syntax

Fortran specification:

```
use armpl_library

real function clanhf(NORM, TRANSR, UPLO, N, A, WORK)
```

C specification:

```
#include "armpl.h"

float clanhf_(const char *norm, const char *transr, const char *uplo,
              const armpl_int_t *n, const armpl_singlecomplex_t *a,
              float *work, ... );
```

Returns

CLANHF = (max(abs(A(i,j))), NORM = 'M' or 'm'

((norm1(A), NORM = '1', 'O' or 'o' ((normI(A), NORM = 'I' or 'i' ((normF(A), NORM = 'F', 'f', 'E' or 'e'

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER

Specifies the value to be returned in CLANHF as described above.

TRANSR Input parameter.

TRANSR is CHARACTER

Specifies whether the RFP format of A is normal or conjugate-transposed format. = 'N': RFP format is Normal = 'C': RFP format is Conjugate-transposed

UPLO Input parameter.

UPLO is CHARACTER

On entry, UPLO specifies whether the RFP matrix A came from an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' RFP A came from an upper triangular matrix

UPLO = 'L' or 'l' RFP A came from a lower triangular matrix

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0. When N = 0, CLANHF is set to zero.

A Input parameter.

A is COMPLEX

A is an array, dimension (N*(N+1)/2);. On entry, the matrix A in RFP Format. RFP Format is described by TRANSR, UPLO and N as follows: If TRANSR='N' then RFP A is (0:N,0:K-1) when N is even; K=N/2. RFP A is (0:N-1,0:K) when N is odd; K=N/2. If TRANSR = 'C' then RFP is the Conjugate-transpose of RFP A as defined when TRANSR = 'N'. The contents of RFP A are defined by UPLO as follows: If UPLO = 'U' the RFP A contains the (N*(N+1)/2) elements of upper packed A either in normal or conjugate-transpose Format. If UPLO = 'L' the RFP A contains the (N*(N+1) /2) elements of lower packed A either in normal or conjugate-transpose Format. The LDA of RFP A is (N+1)/2 when TRANSR = 'C'. When TRANSR is 'N' the LDA is N+1 when N is even and is N when is odd. See the Note below for more details. Unchanged on exit.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (LWORK),. where LWORK >= N when NORM = 'I' or '1' or 'O'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *zlanhf*.

4.17.85 clanhp

clanhp returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a complex hermitian matrix A, supplied in packed form.

Syntax

Fortran specification:

```
use armpl_library

real function clanhp(NORM, UPLO, N, AP, WORK)
```

C specification:

```
#include "armpl.h"

float clanhp_(const char *norm, const char *uplo, const armpl_int_t *n,
              const armpl_singlecomplex_t *ap, float *work, ... );
```

Returns

CLANHP = (max(abs(A(i,j))), NORM = 'M' or 'm'

((norm1(A), NORM = '1', 'O' or 'o' ((normI(A), NORM = 'I' or 'i' ((normF(A), NORM = 'F', 'f', 'E' or 'e'

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in CLANHP as described above.

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the hermitian matrix A is supplied. = 'U': Upper triangular part of A is supplied = 'L': Lower triangular part of A is supplied

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0. When N = 0, CLANHP is set to zero.

AP Input parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). The upper or lower triangle of the hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n. Note that the imaginary parts of the diagonal elements need not be set and are assumed to be zero.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= N when NORM = 'I' or '1' or 'O'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *zlanhp*.

4.17.86 clanhs

clanhs returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a Hessenberg matrix A.

Syntax

Fortran specification:

```
use armpl_library

real function clanhs(NORM, N, A, LDA, WORK)
```

C specification:

```
#include "armpl.h"

float clanhs_(const char *norm, const armpl_int_t *n,
              const armpl_singlecomplex_t *a, const armpl_int_t *lda,
              float *work, ... );
```

Returns

CLANHS = (max(abs(A(i,j))), NORM = 'M' or 'm'

((norm1(A), NORM = '1', 'O' or 'o' ((normI(A), NORM = 'I' or 'i' ((normF(A), NORM = 'F', 'f', 'E' or 'e'

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in CLANHS as described above.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0. When N = 0, CLANHS is set to zero.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The n by n upper Hessenberg matrix A; the part of A below the first sub-diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(N,1).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= N when NORM = 'I'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *dlanhs*, *slanhs* and *zlanhs*.

4.17.87 clanht

clanht returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a complex Hermitian tridiagonal matrix A.

Syntax

Fortran specification:

```
use armpl_library

real function clanht(NORM, N, D, E)
```

C specification:

```
#include "armpl.h"

float clanht_(const char *norm, const armpl_int_t *n, const float *d,
              const armpl_singlecomplex_t *e, ... );
```

Returns

CLANHT = (max(abs(A(i,j))), NORM = 'M' or 'm'

((norm1(A), NORM = '1', 'O' or 'o' ((normI(A), NORM = 'I' or 'i' ((normF(A), NORM = 'F', 'f', 'E' or 'e'

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in CLANHT as described above.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0. When N = 0, CLANHT is set to zero.

D Input parameter.

D is REAL

D is an array, dimension (N). The diagonal elements of A.

E Input parameter.

E is COMPLEX

E is an array, dimension (N-1). The (n-1) sub-diagonal or super-diagonal elements of A.

Related Information

For this routine in other precisions, please see *zlanht*.

4.17.88 clansb

clansb returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of an n by n symmetric band matrix A, with k super-diagonals.

Syntax

Fortran specification:

```
use armpl_library

real function clansb(NORM, UPLO, N, K, AB, LDAB, WORK)
```

C specification:

```
#include "armpl.h"

float clansb_(const char *norm, const char *uplo, const armpl_int_t *n,
              const armpl_int_t *k, const armpl_singlecomplex_t *ab,
              const armpl_int_t *ldab, float *work, ... );
```

Returns

CLANSB = (max(abs(A(i,j))), NORM = ‘M’ or ‘m’

((norm1(A), NORM = ‘1’, ‘O’ or ‘o’ ((normI(A), NORM = ‘I’ or ‘i’ ((normF(A), NORM = ‘F’, ‘f’, ‘E’ or ‘e’

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in CLANSB as described above.

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the band matrix A is supplied. = ‘U’: Upper triangular part is supplied = ‘L’: Lower triangular part is supplied

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0. When N = 0, CLANSB is set to zero.

K Input parameter.

K is INTEGER

The number of super-diagonals or sub-diagonals of the band matrix A. K >= 0.

AB Input parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). The upper or lower triangle of the symmetric band matrix A, stored in the first K+1 rows of AB. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = ‘U’, AB(k+1+i-j,j) = A(i,j) for max(1,j-k)<=i<=j; if UPLO = ‘L’, AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+k).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= K+1.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= N when NORM = 'I' or '1' or 'O'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *dlansb*, *slansb* and *zlansb*.

4.17.89 clansp

clansp returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a complex symmetric matrix A, supplied in packed form.

Syntax

Fortran specification:

```
use armpl_library

real function clansp(NORM, UPLO, N, AP, WORK)
```

C specification:

```
#include "armpl.h"

float clansp_(const char *norm, const char *uplo, const armpl_int_t *n,
              const armpl_singlecomplex_t *ap, float *work, ... );
```

Returns

CLANSP = (max(abs(A(i,j))), NORM = 'M' or 'm'

((norm1(A), NORM = '1', 'O' or 'o' ((normI(A), NORM = 'I' or 'i' ((normF(A), NORM = 'F', 'f', 'E' or 'e'

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in CLANSP as described above.

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is supplied. = 'U': Upper triangular part of A is supplied = 'L': Lower triangular part of A is supplied

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0. When N = 0, CLANSP is set to zero.

AP Input parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). The upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= N when NORM = 'I' or '1' or 'O'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *dlansp*, *slansp* and *zlansp*.

4.17.90 clansy

clansy returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a complex symmetric matrix A.

Syntax

Fortran specification:

```
use armpl_library

real function clansy(NORM, UPLO, N, A, LDA, WORK)
```

C specification:

```
#include "armpl.h"

float clansy_(const char *norm, const char *uplo, const armpl_int_t *n,
              const armpl_singlecomplex_t *a, const armpl_int_t *lda,
              float *work, ... );
```

Returns

CLANSY = (max(abs(A(i,j))), NORM = 'M' or 'm'

((norm1(A), NORM = '1', 'O' or 'o' ((normI(A), NORM = 'I' or 'i' ((normF(A), NORM = 'F', 'f', 'E' or 'e'

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in CLANSY as described above.

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is to be referenced. = 'U': Upper triangular part of A is referenced = 'L': Lower triangular part of A is referenced

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0. When N = 0, CLANSY is set to zero.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The symmetric matrix A. If UPLO = 'U', the leading n by n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n by n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(N,1).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= N when NORM = 'I' or '1' or 'O'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *dlansy*, *slansy* and *zlansy*. It also exists with a native C interface as *LAPACKE_clansy*.

4.17.91 clantb

clantb returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of an n by n triangular band matrix A, with (k + 1) diagonals.

Syntax

Fortran specification:

```
use armpl_library

real function clantb(NORM, UPLO, DIAG, N, K, AB, LDAB, WORK)
```

C specification:

```
#include "armpl.h"

float clantb_(const char *norm, const char *uplo, const char *diag,
              const armpl_int_t *n, const armpl_int_t *k,
              const armpl_singlecomplex_t *ab, const armpl_int_t *ldab,
              float *work, ... );
```

Returns

CLANTB = (max(abs(A(i,j))), NORM = 'M' or 'm'

((norm1(A), NORM = '1', 'O' or 'o' ((normI(A), NORM = 'I' or 'i' ((normF(A), NORM = 'F', 'f', 'E' or 'e'

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in CLANTB as described above.

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the matrix A is upper or lower triangular. = 'U': Upper triangular = 'L': Lower triangular

DIAG Input parameter.

DIAG is CHARACTER*1

Specifies whether or not the matrix A is unit triangular. = 'N': Non-unit triangular = 'U': Unit triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0. When N = 0, CLANTB is set to zero.

K Input parameter.

K is INTEGER

The number of super-diagonals of the matrix A if UPLO = 'U', or the number of sub-diagonals of the matrix A if UPLO = 'L'. K >= 0.

AB Input parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). The upper or lower triangular band matrix A, stored in the first k+1 rows of AB. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(k+1+i-j,j) = A(i,j) for max(1,j-k)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+k). Note that when DIAG = 'U', the elements of the array AB corresponding to the diagonal elements of the matrix A are not referenced, but are assumed to be one.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= K+1.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= N when NORM = 'I'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *dlantb*, *slantb* and *zlantb*.

4.17.92 clantp

`clantp` returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a triangular matrix A, supplied in packed form.

Syntax

Fortran specification:

```
use armpl_library

real function clantp(NORM, UPLO, DIAG, N, AP, WORK)
```

C specification:

```
#include "armpl.h"

float clantp_(const char *norm, const char *uplo, const char *diag,
              const armpl_int_t *n, const armpl_singlecomplex_t *ap,
              float *work, ... );
```

Returns

CLANTP = (max(abs(A(i,j))), NORM = 'M' or 'm'

((norm1(A), NORM = '1', 'O' or 'o' ((normI(A), NORM = 'I' or 'i' ((normF(A), NORM = 'F', 'f', 'E' or 'e'

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

`NORM` is `CHARACTER*1`

Specifies the value to be returned in CLANTP as described above.

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

Specifies whether the matrix A is upper or lower triangular. = 'U': Upper triangular = 'L': Lower triangular

DIAG Input parameter.

`DIAG` is `CHARACTER*1`

Specifies whether or not the matrix A is unit triangular. = 'N': Non-unit triangular = 'U': Unit triangular

N Input parameter.

`N` is `INTEGER`

The order of the matrix A. `N` >= 0. When `N` = 0, CLANTP is set to zero.

AP Input parameter.

`AP` is `COMPLEX`

`AP` is an array, dimension (N*(N+1)/2). The upper or lower triangular matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array `AP` as follows: if `UPLO` = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if `UPLO` = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n. Note that when `DIAG` = 'U',

the elements of the array AP corresponding to the diagonal elements of the matrix A are not referenced, but are assumed to be one.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= N when NORM = 'I'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *dlantp*, *slantp* and *zlantp*.

4.17.93 clantr

clantr returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a trapezoidal or triangular matrix A.

Syntax

Fortran specification:

```
use armpl_library

real function clantr(NORM, UPLO, DIAG, M, N, A, LDA, WORK)
```

C specification:

```
#include "armpl.h"

float clantr_(const char *norm, const char *uplo, const char *diag,
              const armpl_int_t *m, const armpl_int_t *n,
              const armpl_singlecomplex_t *a, const armpl_int_t *lda,
              float *work, ... );
```

Returns

CLANTR = (max(abs(A(i,j))), NORM = 'M' or 'm'

((norm1(A), NORM = '1', 'O' or 'o' ((normI(A), NORM = 'I' or 'i' ((normF(A), NORM = 'F', 'f', 'E' or 'e'

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in CLANTR as described above.

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the matrix A is upper or lower trapezoidal. = 'U': Upper trapezoidal = 'L': Lower trapezoidal Note that A is triangular instead of trapezoidal if M = N.

DIAG Input parameter.

DIAG is CHARACTER*1

Specifies whether or not the matrix A has unit diagonal. = 'N': Non-unit diagonal = 'U': Unit diagonal

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0, and if UPLO = 'U', M <= N. When M = 0, CLANTR is set to zero.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0, and if UPLO = 'L', N <= M. When N = 0, CLANTR is set to zero.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The trapezoidal matrix A (A is triangular if M = N). If UPLO = 'U', the leading m by n upper trapezoidal part of the array A contains the upper trapezoidal matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading m by n lower trapezoidal part of the array A contains the lower trapezoidal matrix, and the strictly upper triangular part of A is not referenced. Note that when DIAG = 'U', the diagonal elements of A are not referenced and are assumed to be one.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(M,1).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= M when NORM = 'I'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *dlantr*, *slantr* and *zlantr*. It also exists with a native C interface as *LAPACKE_clantr*.

4.17.94 clapll

Given two column vectors X and Y, let

```
A = ( X Y ).
```

The subroutine first computes the QR factorization of A = Q*R, and then computes the SVD of the 2-by-2 upper triangular matrix R. The smaller singular value of R is returned in SSMIN, which is used as the measurement of the linear dependency of the vectors X and Y.

Syntax

Fortran specification:

```
use armpl_library

subroutine clapll(N, X, INCX, Y, INCY, SSMIN)
```

C specification:

```
#include "armpl.h"

void clapll_(const armpl_int_t *n, armpl_singlecomplex_t *x,
             const armpl_int_t *incx, armpl_singlecomplex_t *y,
             const armpl_int_t *incy, float *ssmin);
```

Parameters

N Input parameter.

N is INTEGER

The length of the vectors X and Y.

X Input and output parameter.

X is COMPLEX

X is an array, dimension (1+(N-1)*INCX). On entry, X contains the N-vector X. On exit, X is overwritten.

INCX Input parameter.

INCX is INTEGER

The increment between successive elements of X. INCX > 0.

Y Input and output parameter.

Y is COMPLEX

Y is an array, dimension (1+(N-1)*INCY). On entry, Y contains the N-vector Y. On exit, Y is overwritten.

INCY Input parameter.

INCY is INTEGER

The increment between successive elements of Y. INCY > 0.

SSMIN Output parameter.

SSMIN is REAL

The smallest singular value of the N-by-2 matrix A = (X Y).

Related Information

For this routine in other precisions, please see *dlapll*, *slapll* and *zlapll*.

4.17.95 clapmr

clapmr rearranges the rows of the M by N matrix X as specified by the permutation K(1),K(2),...,K(M) of the integers 1,...,M. If FORWRD = .TRUE., forward permutation:

```
X(K(I),*) is moved X(I,*) for I = 1,2,...,M.
```

If FORWRD = .FALSE., backward permutation:

```
X(I,*) is moved to X(K(I),*) for I = 1,2,...,M.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine clapmr(FORWRD, M, N, X, LDX, K)
```

C specification:

```
#include "armpl.h"

void clapmr_(const armpl_int_t *forwrd, const armpl_int_t *m,
             const armpl_int_t *n, armpl_singlecomplex_t *x,
             const armpl_int_t *ldx, armpl_int_t *k);
```

Parameters

FORWRD Input parameter.

FORWRD is LOGICAL

= .TRUE., forward permutation = .FALSE., backward permutation

M Input parameter.

M is INTEGER

The number of rows of the matrix X. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix X. N >= 0.

X Input and output parameter.

X is COMPLEX

X is an array, dimension (LDX, N). On entry, the M by N matrix X. On exit, X contains the permuted matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X, LDX >= MAX(1, M).

K Input and output parameter.

K is INTEGER array, dimension (M)

On entry, K contains the permutation vector. K is used as internal workspace, but reset to its original value on output.

Related Information

For this routine in other precisions, please see *dlapmr*, *slapmr* and *zlapmr*. It also exists with a native C interface as *LAPACKE_clapmr*.

4.17.96 clapmt

clapmt rearranges the columns of the M by N matrix X as specified by the permutation K(1),K(2),...,K(N) of the integers 1,...,N. If FORWRD = .TRUE., forward permutation:

```
X(*,K(J)) is moved X(*,J) for J = 1,2,...,N.
```

If FORWRD = .FALSE., backward permutation:

```
X(*,J) is moved to X(*,K(J)) for J = 1,2,...,N.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine clapmt(FORWRD, M, N, X, LDX, K)
```

C specification:

```
#include "armpl.h"

void clapmt_(const armpl_int_t *forwrd, const armpl_int_t *m,
             const armpl_int_t *n, armpl_singlecomplex_t *x,
             const armpl_int_t *ldx, armpl_int_t *k);
```

Parameters

FORWRD Input parameter.

FORWRD is LOGICAL

= .TRUE., forward permutation = .FALSE., backward permutation

M Input parameter.

M is INTEGER

The number of rows of the matrix X. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix X. N >= 0.

X Input and output parameter.

X is COMPLEX

X is an array, dimension (LDX, N). On entry, the M by N matrix X. On exit, X contains the permuted matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X, LDX >= MAX(1, M).

K Input and output parameter.

K is INTEGER array, dimension (N)

On entry, K contains the permutation vector. K is used as internal workspace, but reset to its original value on output.

Related Information

For this routine in other precisions, please see *dlapmt*, *slapmt* and *zlapmt*. It also exists with a native C interface as *LAPACKE_clapmt*.

4.17.97 claqgb

`claqgb` equilibrates a general M by N band matrix A with KL subdiagonals and KU superdiagonals using the row and scaling factors in the vectors R and C.

Syntax

Fortran specification:

```
use armpl_library

subroutine claqgb(M, N, KL, KU, AB, LDAB, R, C, ROWCND, COLCND, AMAX, EQUED)
```

C specification:

```
#include "armpl.h"

void claqgb_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *kl, const armpl_int_t *ku,
             armpl_singlecomplex_t *ab, const armpl_int_t *ldab,
             const float *r, const float *c, const float *rowcnd,
             const float *colcnd, const float *amax, char *equed, ... );
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

AB Input and output parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(ku+1+i-j,j) = A(i,j) for max(1,j-ku)<=i<=min(m,j+kl)

On exit, the equilibrated matrix, in the same storage format as A. See EQUED for the form of the equilibrated matrix.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDA >= KL+KU+1.

R Input parameter.

R is REAL

R is an array, dimension (M). The row scale factors for A.

C Input parameter.

C is REAL

C is an array, dimension (N). The column scale factors for A.

ROWCND Input parameter.

ROWCND is REAL

Ratio of the smallest R(i) to the largest R(i).

COLCND Input parameter.

COLCND is REAL

Ratio of the smallest C(i) to the largest C(i).

AMAX Input parameter.

AMAX is REAL

Absolute value of largest matrix entry.

EQUED Output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration = 'R': Row equilibration, i.e., A has been premultiplied by diag(R). = 'C': Column equilibration, i.e., A has been postmultiplied by diag(C). = 'B': Both row and column equilibration, i.e., A has been replaced by diag(R) * A * diag(C).

Related Information

For this routine in other precisions, please see *dlaqgb*, *slaqgb* and *zlaqgb*.

4.17.98 claqge

claqge equilibrates a general M by N matrix A using the row and column scaling factors in the vectors R and C.

Syntax

Fortran specification:

```
use armpl_library

subroutine claqge(M, N, A, LDA, R, C, ROWCND, COLCND, AMAX, EQUED)
```

C specification:

```
#include "armpl.h"

void claqge_(const armpl_int_t *m, const armpl_int_t *n,
             armpl_singlecomplex_t *a, const armpl_int_t *lda, const float *r,
             const float *c, const float *rowcnd, const float *colcnd,
             const float *amax, char *equed, ... );
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the M by N matrix A. On exit, the equilibrated matrix. See EQUED for the form of the equilibrated matrix.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(M,1).

R Input parameter.

R is REAL

R is an array, dimension (M). The row scale factors for A.

C Input parameter.

C is REAL

C is an array, dimension (N). The column scale factors for A.

ROWCND Input parameter.

ROWCND is REAL

Ratio of the smallest R(i) to the largest R(i).

COLCND Input parameter.

COLCND is REAL

Ratio of the smallest C(i) to the largest C(i).

AMAX Input parameter.

AMAX is REAL

Absolute value of largest matrix entry.

EQUED Output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration = 'R': Row equilibration, i.e., A has been premultiplied by diag(R). = 'C': Column equilibration, i.e., A has been postmultiplied by diag(C). = 'B': Both row and column equilibration, i.e., A has been replaced by diag(R) * A * diag(C).

Related Information

For this routine in other precisions, please see *dlaqge*, *slaqge* and *zlaqge*.

4.17.99 claqhb

claqhb equilibrates an Hermitian band matrix A using the scaling factors in the vector S.

Syntax

Fortran specification:

```
use armpl_library

subroutine claqhb(UPLO, N, KD, AB, LDAB, S, SCOND, AMAX, EQUED)
```

C specification:

```
#include "armpl.h"

void claqhb_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
             armpl_singlecomplex_t *ab, const armpl_int_t *ldab, float *s,
             const float *scond, const float *amax, char *equed, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored. = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of super-diagonals of the matrix A if UPLO = 'U', or the number of sub-diagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, if INFO = 0, the triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H of the band matrix A, in the same storage format as A.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

S Output parameter.

S is REAL

S is an array, dimension (N). The scale factors for A.

SCOND Input parameter.

SCOND is REAL

Ratio of the smallest S(i) to the largest S(i).

AMAX Input parameter.

AMAX is REAL

Absolute value of largest matrix entry.

EQUED Output parameter.

EQUED is CHARACTER*1

Specifies whether or not equilibration was done. = 'N': No equilibration. = 'Y': Equilibration was done, i.e., A has been replaced by diag(S) * A * diag(S).

Related Information

For this routine in other precisions, please see *zlaqhb*.

4.17.100 claqhe

claqhe equilibrates a Hermitian matrix A using the scaling factors in the vector S.

Syntax

Fortran specification:

```
use armpl_library

subroutine claqhe(UPLO, N, A, LDA, S, SCOND, AMAX, EQUED)
```

C specification:

```
#include "armpl.h"

void claqhe_(const char *uplo, const armpl_int_t *n, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, const float *s, const float *scond,
             const float *amax, char *equed, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the Hermitian matrix A is stored. = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading n by n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of

A is not referenced. If UPLO = 'L', the leading n by n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if EQUED = 'Y', the equilibrated matrix: diag(S) * A * diag(S).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(N,1).

S Input parameter.

S is REAL

S is an array, dimension (N). The scale factors for A.

SCOND Input parameter.

SCOND is REAL

Ratio of the smallest S(i) to the largest S(i).

AMAX Input parameter.

AMAX is REAL

Absolute value of largest matrix entry.

EQUED Output parameter.

EQUED is CHARACTER*1

Specifies whether or not equilibration was done. = 'N': No equilibration. = 'Y': Equilibration was done, i.e., A has been replaced by diag(S) * A * diag(S).

Related Information

For this routine in other precisions, please see *zlaqhe*.

4.17.101 claqhp

claqhp equilibrates a Hermitian matrix A using the scaling factors in the vector S.

Syntax

Fortran specification:

```
use armpl_library

subroutine claqhp(UPLO, N, AP, S, SCOND, AMAX, EQUED)
```

C specification:

```
#include "armpl.h"

void claqhp_(const char *uplo, const armpl_int_t *n,
             armpl_singlecomplex_t *ap, const float *s, const float *scond,
             const float *amax, char *equed, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the Hermitian matrix A is stored. = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the Hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

On exit, the equilibrated matrix: diag(S) * A * diag(S), in the same storage format as A.

S Input parameter.

S is REAL

S is an array, dimension (N). The scale factors for A.

SCOND Input parameter.

SCOND is REAL

Ratio of the smallest S(i) to the largest S(i).

AMAX Input parameter.

AMAX is REAL

Absolute value of largest matrix entry.

EQUED Output parameter.

EQUED is CHARACTER*1

Specifies whether or not equilibration was done. = 'N': No equilibration. = 'Y': Equilibration was done, i.e., A has been replaced by diag(S) * A * diag(S).

Related Information

For this routine in other precisions, please see *zlaqhp*.

4.17.102 claqp2

claqp2 computes a QR factorization with column pivoting of the block A(OFFSET+1:M,1:N). The block A(1:OFFSET,1:N) is accordingly pivoted, but not factorized.

Syntax

Fortran specification:

```
use armpl_library

subroutine claqp2(M, N, OFFSET, A, LDA, JPVT, TAU, VN1, VN2, WORK)
```

C specification:

```
#include "armpl.h"

void claqp2_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *offset, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *jpvt,
             armpl_singlecomplex_t *tau, float *vn1, float *vn2,
             armpl_singlecomplex_t *work);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

OFFSET Input parameter.

OFFSET is INTEGER

The number of rows of the matrix A that must be pivoted but no factorized. OFFSET >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the upper triangle of block A(OFFSET+1:M,1:N) is the triangular factor obtained; the elements in block A(OFFSET+1:M,1:N) below the diagonal, together with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors. Block A(1:OFFSET,1:N) has been accordingly pivoted, but no factorized.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

JPVT Input and output parameter.

JPVT is INTEGER array, dimension (N)

On entry, if JPVT(i) .ne. 0, the i-th column of A is permuted to the front of A*P (a leading column); if JPVT(i) = 0, the i-th column of A is a free column. On exit, if JPVT(i) = k, then the i-th column of A*P was the k-th column of A.

TAU Output parameter.

TAU is COMPLEX

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors.

VN1 Input and output parameter.

VN1 is REAL

VN1 is an array, dimension (N). The vector with the partial column norms.

VN2 Input and output parameter.

VN2 is REAL

VN2 is an array, dimension (N). The vector with the exact column norms.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (N) .

Related Information

For this routine in other precisions, please see *dlaqp2*, *slaqp2* and *zlaqp2*.

4.17.103 claqps

claqps computes a step of QR factorization with column pivoting of a complex M-by-N matrix A by using Blas-3. It tries to factorize NB columns from A starting from the row OFFSET+1, and updates all of the matrix with Blas-3 xGEMM.

In some cases, due to catastrophic cancellations, it cannot factorize NB columns. Hence, the actual number of factorized columns is returned in KB.

Block A(1:OFFSET,1:N) is accordingly pivoted, but not factorized.

Syntax

Fortran specification:

```
use armpl_library

subroutine claqps(M, N, OFFSET, NB, KB, A, LDA, JPVT, TAU, VN1, VN2, AUXV, F,
                  LDF)
```

C specification:

```
#include "armpl.h"

void claqps_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *offset, const armpl_int_t *nb,
             armpl_int_t *kb, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *jpvt,
             armpl_singlecomplex_t *tau, float *vn1, float *vn2,
             armpl_singlecomplex_t *auxv, armpl_singlecomplex_t *f,
             const armpl_int_t *ldf);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0

OFFSET Input parameter.

OFFSET is INTEGER

The number of rows of A that have been factorized in previous steps.

NB Input parameter.

NB is INTEGER

The number of columns to factorize.

KB Output parameter.

KB is INTEGER

The number of columns actually factorized.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, block A(OFFSET+1:M,1:KB) is the triangular factor obtained and block A(1:OFFSET,1:N) has been accordingly pivoted, but no factorized. The rest of the matrix, block A(OFFSET+1:M,KB+1:N) has been updated.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

JPVT Input and output parameter.

JPVT is INTEGER array, dimension (N)

JPVT(I) = K <==> Column K of the full matrix A has been permuted into position I in AP.

TAU Output parameter.

TAU is COMPLEX

TAU is an array, dimension (KB). The scalar factors of the elementary reflectors.

VN1 Input and output parameter.

VN1 is REAL

VN1 is an array, dimension (N). The vector with the partial column norms.

VN2 Input and output parameter.

VN2 is REAL

VN2 is an array, dimension (N). The vector with the exact column norms.

AUXV Input and output parameter.

AUXV is COMPLEX

AUXV is an array, dimension (NB). Auxiliar vector.

F Input and output parameter.

F is COMPLEX

F is an array, dimension (LDF, NB). Matrix $F^H = L * Y^H * A$.

LDF Input parameter.

LDF is INTEGER

The leading dimension of the array F. LDF >= max(1, N).

Related Information

For this routine in other precisions, please see *dlaqps*, *slaqps* and *zlaqps*.

4.17.104 claqr0

```
CLAQR0 computes the eigenvalues of a Hessenberg matrix H
   and, optionally, the matrices T and Z from the Schur decomposition
   H = Z T Z**H, where T is an upper triangular matrix (the
   Schur form), and Z is the unitary matrix of Schur vectors.

   Optionally Z may be postmultiplied into an input unitary
   matrix Q so that this routine can give the Schur factorization
   of a matrix A which has been reduced to the Hessenberg form H
   by the unitary matrix Q:  A = Q*H*Q**H = (QZ)*H*(QZ)**H.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine claqr0(WANTT, WANTZ, N, ILO, IHI, H, LDH, W, ILOZ, IHIZ, Z, LDZ,
                  WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void claqr0_(const armpl_int_t *wantt, const armpl_int_t *wantz,
             const armpl_int_t *n, const armpl_int_t *ilo,
             const armpl_int_t *ihi, armpl_singlecomplex_t *h,
             const armpl_int_t *ldh, armpl_singlecomplex_t *w,
             const armpl_int_t *iloz, const armpl_int_t *ihiz,
             armpl_singlecomplex_t *z, const armpl_int_t *ldz,
             armpl_singlecomplex_t *work, const armpl_int_t *lwork,
             armpl_int_t *info);
```

Parameters

WANTT Input parameter.

WANTT is LOGICAL

= .TRUE. : the full Schur form T is required; = .FALSE.: only eigenvalues are required.

WANTZ Input parameter.

WANTZ is LOGICAL

= .TRUE. : the matrix of Schur vectors Z is required; = .FALSE.: Schur vectors are not required.

N Input parameter.

N is INTEGER

The order of the matrix H. N .GE. 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

It is assumed that H is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N and, if ILO.GT.1, H(ILO,ILO-1) is zero. ILO and IHI are normally set by a previous call to CGEBAL, and then passed to

CGEHRD when the matrix output by CGEBAL is reduced to Hessenberg form. Otherwise, ILO and IHI should be set to 1 and N, respectively. If N.GT.0, then 1.LE.ILO.LE.IHI.LE.N. If N = 0, then ILO = 1 and IHI = 0.

H Input and output parameter.

H is COMPLEX

H is an array, dimension (LDH, N). On entry, the upper Hessenberg matrix H. On exit, if INFO = 0 and WANTT is .TRUE., then H contains the upper triangular matrix T from the Schur decomposition (the Schur form). If INFO = 0 and WANT is .FALSE., then the contents of H are unspecified on exit. (The output value of H when INFO.GT.0 is given under the description of INFO below.)

This subroutine may explicitly set H(i,j) = 0 for i.GT.j and j = 1, 2, ... ILO-1 or j = IHI+1, IHI+2, ... N.

LDH Input parameter.

LDH is INTEGER

The leading dimension of the array H. LDH .GE. max(1, N).

W Output parameter.

W is COMPLEX

W is an array, dimension (N). The computed eigenvalues of H(ILO:IHI,ILO:IHI) are stored in W(ILO:IHI). If WANTT is .TRUE., then the eigenvalues are stored in the same order as on the diagonal of the Schur form returned in H, with W(i) = H(i,i).

ILOZ Input parameter.

ILOZ is INTEGER

IHIZ Input parameter.

IHIZ is INTEGER

Specify the rows of Z to which transformations must be applied if WANTZ is .TRUE.. 1 .LE. ILOZ .LE. ILO; IHI .LE. IHIZ .LE. N.

Z Input and output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, IHI). If WANTZ is .FALSE., then Z is not referenced. If WANTZ is .TRUE., then Z(ILO:IHI,ILOZ:IHIZ) is replaced by Z(ILO:IHI,ILOZ:IHIZ)*U where U is the orthogonal Schur factor of H(ILO:IHI,ILO:IHI). (The output value of Z when INFO.GT.0 is given under the description of INFO below.)

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. if WANTZ is .TRUE. then LDZ.GE.MAX(1, IHIZ). Otherwize, LDZ.GE.1.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension LWORK. On exit, if LWORK = -1, WORK(1) returns an estimate of the optimal value for LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK .GE. max(1, N) is sufficient, but LWORK typically as large as 6* N may be required for optimal performance. A workspace query to determine the optimal workspace size is recommended.

If LWORK = -1, then CLAQR0 does a workspace query. In this case, CLAQR0 checks the input parameters and estimates the optimal workspace size for the given values of N, ILO and IHI. The estimate is returned in WORK(1). No error message related to LWORK is issued by XERBLA. Neither H nor Z are accessed.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit .GT. 0: if INFO = i, CLAQR0 failed to compute all of the eigenvalues. Elements 1:ilo-1 and i+1:n of WR and WI contain those eigenvalues which have been successfully computed. (Failures are rare.)

If INFO .GT. 0 and WANT is .FALSE., then on exit, the remaining unconverged eigenvalues are the eigenvalues of the upper Hessenberg matrix rows and columns ILO through INFO of the final, output value of H.

If INFO .GT. 0 and WANTT is .TRUE., then on exit

(*) (initial value of H)*U = U*(final value of H)

where U is a unitary matrix. The final value of H is upper Hessenberg and triangular in rows and columns INFO+1 through IHI.

If INFO .GT. 0 and WANTZ is .TRUE., then on exit

(final value of Z(ILO:IHI,ILOZ:IHIZ) = (initial value of Z(ILO:IHI,ILOZ:IHIZ)*U

where U is the unitary matrix in (*) (regard- less of the value of WANTT.)

If INFO .GT. 0 and WANTZ is .FALSE., then Z is not accessed.

Related Information

For this routine in other precisions, please see *dlaqr0*, *slaqr0* and *zlaqr0*.

4.17.105 claqr1

```
Given a 2-by-2 or 3-by-3 matrix H, CLAQR1 sets v to a
     scalar multiple of the first column of the product

     (*)  K = (H - s1*I)*(H - s2*I)

     scaling to avoid overflows and most underflows.

     This is useful for starting double implicit shift bulges
     in the QR algorithm.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine claqr1(N, H, LDH, S1, S2, V)
```

C specification:

```
#include "armpl.h"

void claqr1_(const armpl_int_t *n, const armpl_singlecomplex_t *h,
             const armpl_int_t *ldh, const armpl_singlecomplex_t *s1,
             armpl_singlecomplex_t *s2, armpl_singlecomplex_t *v);
```

Parameters

N Input parameter.

N is INTEGER

Order of the matrix H. N must be either 2 or 3.

H Input parameter.

H is COMPLEX

H is an array, dimension (LDH, N). The 2-by-2 or 3-by-3 matrix H in (*).

LDH Input parameter.

LDH is INTEGER

The leading dimension of H as declared in the calling procedure. LDH.GE.N

S1 Input parameter.

S1 is COMPLEX

S2 Input parameter.

S2 is COMPLEX

S1 and S2 are the shifts defining K in (*) above.

V Output parameter.

V is COMPLEX

V is an array, dimension (N). A scalar multiple of the first column of the matrix K in (*).

Related Information

For this routine in other precisions, please see *dlaqr1*, *slaqr1* and *zlaqr1*.

4.17.106 claqr2

```
CLAQR2 is identical to CLAQR3 except that it avoids
   recursion by calling CLAHQR instead of CLAQR4.

   Aggressive early deflation:

   This subroutine accepts as input an upper Hessenberg matrix
   H and performs an unitary similarity transformation
   designed to detect and deflate fully converged eigenvalues from
   a trailing principal submatrix.  On output H has been over-
   written by a new Hessenberg matrix that is a perturbation of
   an unitary similarity transformation of H.  It is to be
   hoped that the final version of H has many zero subdiagonal
   entries.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine claqr2(WANTT, WANTZ, N, KTOP, KBOT, NW, H, LDH, ILOZ, IHIZ, Z, LDZ,
                  NS, ND, SH, V, LDV, NH, T, LDT, NV, WV, LDWV, WORK, LWORK)
```

C specification:

```
#include "armpl.h"

void claqr2_(const armpl_int_t *wantt, const armpl_int_t *wantz,
             const armpl_int_t *n, const armpl_int_t *ktop,
             const armpl_int_t *kbot, const armpl_int_t *nw,
             armpl_singlecomplex_t *h, const armpl_int_t *ldh,
             const armpl_int_t *iloz, const armpl_int_t *ihiz,
             armpl_singlecomplex_t *z, const armpl_int_t *ldz,
             armpl_int_t *ns, armpl_int_t *nd, armpl_singlecomplex_t *sh,
             armpl_singlecomplex_t *v, const armpl_int_t *ldv,
             const armpl_int_t *nh, armpl_singlecomplex_t *t,
             const armpl_int_t *ldt, const armpl_int_t *nv,
             armpl_singlecomplex_t *wv, const armpl_int_t *ldwv,
             armpl_singlecomplex_t *work, const armpl_int_t *lwork);
```

Parameters

WANTT Input parameter.

WANTT is LOGICAL

If .TRUE., then the Hessenberg matrix H is fully updated so that the triangular Schur factor may be computed (in cooperation with the calling subroutine). If .FALSE., then only enough of H is updated to preserve the eigenvalues.

WANTZ Input parameter.

WANTZ is LOGICAL

If .TRUE., then the unitary matrix Z is updated so so that the unitary Schur factor may be computed (in cooperation with the calling subroutine). If .FALSE., then Z is not referenced.

N Input parameter.

N is INTEGER

The order of the matrix H and (if WANTZ is .TRUE.) the order of the unitary matrix Z.

KTOP Input parameter.

KTOP is INTEGER

It is assumed that either KTOP = 1 or H(KTOP,KTOP-1)=0. KBOT and KTOP together determine an isolated block along the diagonal of the Hessenberg matrix.

KBOT Input parameter.

KBOT is INTEGER

It is assumed without a check that either KBOT = N or H(KBOT+1, KBOT)=0. KBOT and KTOP together determine an isolated block along the diagonal of the Hessenberg matrix.

NW Input parameter.

NW is INTEGER

Deflation window size. 1 .LE. NW .LE. (KBOT-KTOP+1).

H Input and output parameter.

H is COMPLEX

H is an array, dimension (LDH, N). On input the initial N-by-N section of H stores the Hessenberg matrix undergoing aggressive early deflation. On output H has been transformed by a unitary similarity transformation, perturbed, and the returned to Hessenberg form that (it is to be hoped) has some zero subdiagonal entries.

LDH Input parameter.

LDH is INTEGER

Leading dimension of H just as declared in the calling subroutine. N .LE. LDH

ILOZ Input parameter.

ILOZ is INTEGER

IHIZ Input parameter.

IHIZ is INTEGER

Specify the rows of Z to which transformations must be applied if WANTZ is .TRUE.. 1 .LE. ILOZ .LE. IHIZ .LE. N.

Z Input and output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, N). IF WANTZ is .TRUE., then on output, the unitary similarity transformation mentioned above has been accumulated into Z(ILOZ:IHIZ,ILOZ:IHIZ) from the right. If WANTZ is .FALSE., then Z is unreferenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of Z just as declared in the calling subroutine. 1 .LE. LDZ.

NS Output parameter.

NS is INTEGER

The number of unconverged (ie approximate) eigenvalues returned in SR and SI that may be used as shifts by the calling subroutine.

ND Output parameter.

ND is INTEGER

The number of converged eigenvalues uncovered by this subroutine.

SH Output parameter.

SH is COMPLEX

SH is an array, dimension (KBOT). On output, approximate eigenvalues that may be used for shifts are stored in SH(KBOT-ND-NS+1) through SR(KBOT-ND). Converged eigenvalues are stored in SH(KBOT-ND+1) through SH(KBOT).

V Output parameter.

V is COMPLEX

V is an array, dimension (LDV, NW). An NW-by-NW work array.

LDV Input parameter.

LDV is INTEGER

The leading dimension of V just as declared in the calling subroutine. NW .LE. LDV

NH Input parameter.

NH is INTEGER

The number of columns of T. NH.GE.NW.

T Output parameter.

T is COMPLEX

T is an array, dimension (LDT, NW) .

LDT Input parameter.

LDT is INTEGER

The leading dimension of T just as declared in the calling subroutine. NW .LE. LDT

NV Input parameter.

NV is INTEGER

The number of rows of work array WV available for workspace. NV.GE.NW.

WV Output parameter.

WV is COMPLEX

WV is an array, dimension (LDWV, NW) .

LDWV Input parameter.

LDWV is INTEGER

The leading dimension of W just as declared in the calling subroutine. NW .LE. LDV

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (LWORK). On exit, WORK(1) is set to an estimate of the optimal value of LWORK for the given values of N, NW, KTOP and KBOT.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the work array WORK. LWORK = 2*NW suffices, but greater efficiency may result from larger values of LWORK.

If LWORK = -1, then a workspace query is assumed; CLAQR2 only estimates the optimal workspace size for the given values of N, NW, KTOP and KBOT. The estimate is returned in WORK(1). No error message related to LWORK is issued by XERBLA. Neither H nor Z are accessed.

Related Information

For this routine in other precisions, please see *dlaqr2*, *slaqr2* and *zlaqr2*.

4.17.107 claqr3

```
Aggressive early deflation:

   CLAQR3 accepts as input an upper Hessenberg matrix
   H and performs an unitary similarity transformation
   designed to detect and deflate fully converged eigenvalues from
   a trailing principal submatrix.  On output H has been over-
   written by a new Hessenberg matrix that is a perturbation of
   an unitary similarity transformation of H.  It is to be
   hoped that the final version of H has many zero subdiagonal
   entries.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine claqr3(WANTT, WANTZ, N, KTOP, KBOT, NW, H, LDH, ILOZ, IHIZ, Z, LDZ,
                  NS, ND, SH, V, LDV, NH, T, LDT, NV, WV, LDWV, WORK, LWORK)
```

C specification:

```
#include "armpl.h"

void claqr3_(const armpl_int_t *wantt, const armpl_int_t *wantz,
             const armpl_int_t *n, const armpl_int_t *ktop,
             const armpl_int_t *kbot, const armpl_int_t *nw,
             armpl_singlecomplex_t *h, const armpl_int_t *ldh,
             const armpl_int_t *iloz, const armpl_int_t *ihiz,
             armpl_singlecomplex_t *z, const armpl_int_t *ldz,
             armpl_int_t *ns, armpl_int_t *nd, armpl_singlecomplex_t *sh,
             armpl_singlecomplex_t *v, const armpl_int_t *ldv,
             const armpl_int_t *nh, armpl_singlecomplex_t *t,
             const armpl_int_t *ldt, const armpl_int_t *nv,
             armpl_singlecomplex_t *wv, const armpl_int_t *ldwv,
             armpl_singlecomplex_t *work, const armpl_int_t *lwork);
```

Parameters

WANTT Input parameter.

WANTT is LOGICAL

If .TRUE., then the Hessenberg matrix H is fully updated so that the triangular Schur factor may be computed (in cooperation with the calling subroutine). If .FALSE., then only enough of H is updated to preserve the eigenvalues.

WANTZ Input parameter.

WANTZ is LOGICAL

If .TRUE., then the unitary matrix Z is updated so so that the unitary Schur factor may be computed (in cooperation with the calling subroutine). If .FALSE., then Z is not referenced.

N Input parameter.

N is INTEGER

The order of the matrix H and (if WANTZ is .TRUE.) the order of the unitary matrix Z.

KTOP Input parameter.

KTOP is INTEGER

It is assumed that either KTOP = 1 or H(KTOP,KTOP-1)=0. KBOT and KTOP together determine an isolated block along the diagonal of the Hessenberg matrix.

KBOT Input parameter.

KBOT is INTEGER

It is assumed without a check that either KBOT = N or H(KBOT+1, KBOT)=0. KBOT and KTOP together determine an isolated block along the diagonal of the Hessenberg matrix.

NW Input parameter.

NW is INTEGER

Deflation window size. 1 .LE. NW .LE. (KBOT-KTOP+1).

H Input and output parameter.

H is COMPLEX

H is an array, dimension (LDH, N). On input the initial N-by-N section of H stores the Hessenberg matrix undergoing aggressive early deflation. On output H has been transformed by a unitary similarity transformation, perturbed, and the returned to Hessenberg form that (it is to be hoped) has some zero subdiagonal entries.

LDH Input parameter.

LDH is INTEGER

Leading dimension of H just as declared in the calling subroutine. N .LE. LDH

ILOZ Input parameter.

ILOZ is INTEGER

IHIZ Input parameter.

IHIZ is INTEGER

Specify the rows of Z to which transformations must be applied if WANTZ is .TRUE.. 1 .LE. ILOZ .LE. IHIZ .LE. N.

Z Input and output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, N). IF WANTZ is .TRUE., then on output, the unitary similarity transformation mentioned above has been accumulated into Z(ILOZ:IHIZ,ILOZ:IHIZ) from the right. If WANTZ is .FALSE., then Z is unreferenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of Z just as declared in the calling subroutine. 1 .LE. LDZ.

NS Output parameter.

NS is INTEGER

The number of unconverged (ie approximate) eigenvalues returned in SR and SI that may be used as shifts by the calling subroutine.

ND Output parameter.

ND is INTEGER

The number of converged eigenvalues uncovered by this subroutine.

SH Output parameter.

SH is COMPLEX

SH is an array, dimension (KBOT). On output, approximate eigenvalues that may be used for shifts are stored in SH(KBOT-ND-NS+1) through SR(KBOT-ND). Converged eigenvalues are stored in SH(KBOT-ND+1) through SH(KBOT).

V Output parameter.

V is COMPLEX

V is an array, dimension (LDV, NW). An NW-by-NW work array.

LDV Input parameter.

LDV is INTEGER

The leading dimension of V just as declared in the calling subroutine. NW .LE. LDV

NH Input parameter.

NH is INTEGER

The number of columns of T. NH.GE.NW.

T Output parameter.

T is COMPLEX

T is an array, dimension (LDT, NW) .

LDT Input parameter.

LDT is INTEGER

The leading dimension of T just as declared in the calling subroutine. NW .LE. LDT

NV Input parameter.

NV is INTEGER

The number of rows of work array WV available for workspace. NV.GE.NW.

WV Output parameter.

WV is COMPLEX

WV is an array, dimension (LDWV, NW) .

LDWV Input parameter.

LDWV is INTEGER

The leading dimension of W just as declared in the calling subroutine. NW .LE. LDV

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (LWORK). On exit, WORK(1) is set to an estimate of the optimal value of LWORK for the given values of N, NW, KTOP and KBOT.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the work array WORK. LWORK = 2*NW suffices, but greater efficiency may result from larger values of LWORK.

If LWORK = -1, then a workspace query is assumed; CLAQR3 only estimates the optimal workspace size for the given values of N, NW, KTOP and KBOT. The estimate is returned in WORK(1). No error message related to LWORK is issued by XERBLA. Neither H nor Z are accessed.

Related Information

For this routine in other precisions, please see *dlaqr3*, *slaqr3* and *zlaqr3*.

4.17.108 claqr4

```
CLAQR4 implements one level of recursion for CLAQR0.
   It is a complete implementation of the small bulge multi-shift
   QR algorithm.  It may be called by CLAQR0 and, for large enough
   deflation window size, it may be called by CLAQR3.  This
   subroutine is identical to CLAQR0 except that it calls CLAQR2
   instead of CLAQR3.

   CLAQR4 computes the eigenvalues of a Hessenberg matrix H
```

(continues on next page)

(continued from previous page)

```
   and, optionally, the matrices T and Z from the Schur decomposition
   H = Z T Z**H, where T is an upper triangular matrix (the
   Schur form), and Z is the unitary matrix of Schur vectors.

   Optionally Z may be postmultiplied into an input unitary
   matrix Q so that this routine can give the Schur factorization
   of a matrix A which has been reduced to the Hessenberg form H
   by the unitary matrix Q:  A = Q*H*Q**H = (QZ)*H*(QZ)**H.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine claqr4(WANTT, WANTZ, N, ILO, IHI, H, LDH, W, ILOZ, IHIZ, Z, LDZ,
                  WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void claqr4_(const armpl_int_t *wantt, const armpl_int_t *wantz,
             const armpl_int_t *n, const armpl_int_t *ilo,
             const armpl_int_t *ihi, armpl_singlecomplex_t *h,
             const armpl_int_t *ldh, armpl_singlecomplex_t *w,
             armpl_int_t *iloz, armpl_int_t *ihiz, armpl_singlecomplex_t *z,
             const armpl_int_t *ldz, armpl_singlecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

WANTT Input parameter.

WANTT is LOGICAL

= .TRUE. : the full Schur form T is required; = .FALSE.: only eigenvalues are required.

WANTZ Input parameter.

WANTZ is LOGICAL

= .TRUE. : the matrix of Schur vectors Z is required; = .FALSE.: Schur vectors are not required.

N Input parameter.

N is INTEGER

The order of the matrix H. N .GE. 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

It is assumed that H is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N and, if ILO.GT.1, H(ILO,ILO-1) is zero. ILO and IHI are normally set by a previous call to CGEBAL, and then passed to CGEHRD when the matrix output by CGEBAL is reduced to Hessenberg form. Otherwise, ILO and IHI should be set to 1 and N, respectively. If N.GT.0, then 1.LE.ILO.LE.IHI.LE.N. If N = 0, then ILO = 1 and IHI = 0.

H Input and output parameter.

H is COMPLEX

H is an array, dimension (LDH, N). On entry, the upper Hessenberg matrix H. On exit, if INFO = 0 and WANTT is .TRUE., then H contains the upper triangular matrix T from the Schur decomposition (the Schur form). If INFO = 0 and WANT is .FALSE., then the contents of H are unspecified on exit. (The output value of H when INFO.GT.0 is given under the description of INFO below.)

This subroutine may explicitly set H(i,j) = 0 for i.GT.j and j = 1, 2, ... ILO-1 or j = IHI+1, IHI+2, ... N.

LDH Input parameter.

LDH is INTEGER

The leading dimension of the array H. LDH .GE. max(1, N).

W Output parameter.

W is COMPLEX

W is an array, dimension (N). The computed eigenvalues of H(ILO:IHI,ILO:IHI) are stored in W(ILO:IHI). If WANTT is .TRUE., then the eigenvalues are stored in the same order as on the diagonal of the Schur form returned in H, with W(i) = H(i,i).

ILOZ Input parameter.

ILOZ is INTEGER

IHIZ Input parameter.

IHIZ is INTEGER

Specify the rows of Z to which transformations must be applied if WANTZ is .TRUE.. 1 .LE. ILOZ .LE. ILO; IHI .LE. IHIZ .LE. N.

Z Input and output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, IHI). If WANTZ is .FALSE., then Z is not referenced. If WANTZ is .TRUE., then Z(ILO:IHI,ILOZ:IHIZ) is replaced by Z(ILO:IHI,ILOZ:IHIZ)*U where U is the orthogonal Schur factor of H(ILO:IHI,ILO:IHI). (The output value of Z when INFO.GT.0 is given under the description of INFO below.)

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. if WANTZ is .TRUE. then LDZ.GE.MAX(1, IHIZ). Otherwize, LDZ.GE.1.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension LWORK. On exit, if LWORK = -1, WORK(1) returns an estimate of the optimal value for LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK .GE. max(1, N) is sufficient, but LWORK typically as large as 6* N may be required for optimal performance. A workspace query to determine the optimal workspace size is recommended.

If LWORK = -1, then CLAQR4 does a workspace query. In this case, CLAQR4 checks the input parameters and estimates the optimal workspace size for the given values of N, ILO and IHI. The estimate is returned in WORK(1). No error message related to LWORK is issued by XERBLA. Neither H nor Z are accessed.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit .GT. 0: if INFO = i, CLAQR4 failed to compute all of the eigenvalues. Elements 1:ilo-1 and i+1:n of WR and WI contain those eigenvalues which have been successfully computed. (Failures are rare.)

If INFO .GT. 0 and WANT is .FALSE., then on exit, the remaining unconverged eigenvalues are the eigenvalues of the upper Hessenberg matrix rows and columns ILO through INFO of the final, output value of H.

If INFO .GT. 0 and WANTT is .TRUE., then on exit

(*) (initial value of H)*U = U*(final value of H)

where U is a unitary matrix. The final value of H is upper Hessenberg and triangular in rows and columns INFO+1 through IHI.

If INFO .GT. 0 and WANTZ is .TRUE., then on exit

(final value of Z(ILO:IHI,ILOZ:IHIZ) = (initial value of Z(ILO:IHI,ILOZ:IHIZ)*U

where U is the unitary matrix in (*) (regard- less of the value of WANTT.)

If INFO .GT. 0 and WANTZ is .FALSE., then Z is not accessed.

Related Information

For this routine in other precisions, please see *dlaqr4*, *slaqr4* and *zlaqr4*.

4.17.109 claqr5

```
CLAQR5 called by CLAQR0 performs a
   single small-bulge multi-shift QR sweep.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine claqr5(WANTT, WANTZ, KACC22, N, KTOP, KBOT, NSHFTS, S, H, LDH,
                  ILOZ, IHIZ, Z, LDZ, V, LDV, U, LDU, NV, WV, LDWV, NH, WH,
                  LDWH)
```

C specification:

```
#include "armpl.h"

void claqr5_(const armpl_int_t *wantt, const armpl_int_t *wantz,
             const armpl_int_t *kacc22, const armpl_int_t *n,
             const armpl_int_t *ktop, const armpl_int_t *kbot,
             const armpl_int_t *nshfts, armpl_singlecomplex_t *s,
             armpl_singlecomplex_t *h, const armpl_int_t *ldh,
             const armpl_int_t *iloz, const armpl_int_t *ihiz,
             armpl_singlecomplex_t *z, const armpl_int_t *ldz,
             armpl_singlecomplex_t *v, const armpl_int_t *ldv,
             armpl_singlecomplex_t *u, const armpl_int_t *ldu,
             const armpl_int_t *nv, armpl_singlecomplex_t *wv,
             const armpl_int_t *ldwv, const armpl_int_t *nh,
             armpl_singlecomplex_t *wh, const armpl_int_t *ldwh);
```

Parameters

WANTT Input parameter.

WANTT is LOGICAL

WANTT = .true. if the triangular Schur factor is being computed. WANTT is set to .false. otherwise.

WANTZ Input parameter.

WANTZ is LOGICAL

WANTZ = .true. if the unitary Schur factor is being computed. WANTZ is set to .false. otherwise.

KACC22 Input parameter.

KACC22 is INTEGER with value 0, 1, or 2.

Specifies the computation mode of far-from-diagonal orthogonal updates. = 0: CLAQR5 does not accumulate reflections and does not use matrix-matrix multiply to update far-from-diagonal matrix entries. = 1: CLAQR5 accumulates reflections and uses matrix-matrix multiply to update the far-from-diagonal matrix entries. = 2: CLAQR5 accumulates reflections, uses matrix-matrix multiply to update the far-from-diagonal matrix entries, and takes advantage of 2-by-2 block structure during matrix multiplies.

N Input parameter.

N is INTEGER

N is the order of the Hessenberg matrix H upon which this subroutine operates.

KTOP Input parameter.

KTOP is INTEGER

KBOT Input parameter.

KBOT is INTEGER

These are the first and last rows and columns of an isolated diagonal block upon which the QR sweep is to be applied. It is assumed without a check that either KTOP = 1 or H(KTOP,KTOP-1) = 0 and either KBOT = N or H(KBOT+1, KBOT) = 0.

NSHFTS Input parameter.

NSHFTS is INTEGER

NSHFTS gives the number of simultaneous shifts. NSHFTS must be positive and even.

S Input and output parameter.

S is COMPLEX

S is an array, dimension (NSHFTS). S contains the shifts of origin that define the multi- shift QR sweep. On output S may be reordered.

H Input and output parameter.

H is COMPLEX

H is an array, dimension (LDH, N). On input H contains a Hessenberg matrix. On output a multi-shift QR sweep with shifts SR(J)+i*SI(J) is applied to the isolated diagonal block in rows and columns KTOP through KBOT.

LDH Input parameter.

LDH is INTEGER

LDH is the leading dimension of H just as declared in the calling procedure. LDH.GE.MAX(1, N).

ILOZ Input parameter.

ILOZ is INTEGER

IHIZ Input parameter.

IHIZ is INTEGER

Specify the rows of Z to which transformations must be applied if WANTZ is .TRUE.. 1 .LE. ILOZ .LE. IHIZ .LE. N

Z Input and output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, IHIZ). If WANTZ = .TRUE., then the QR Sweep unitary similarity transformation is accumulated into Z(ILOZ:IHIZ,ILOZ:IHIZ) from the right. If WANTZ = .FALSE., then Z is unreferenced.

LDZ Input parameter.

LDZ is INTEGER

LDA is the leading dimension of Z just as declared in the calling procedure. LDZ.GE.N.

V Output parameter.

V is COMPLEX

V is an array, dimension (LDV,NSHFTS/2) .

LDV Input parameter.

LDV is INTEGER

LDV is the leading dimension of V as declared in the calling procedure. LDV.GE.3.

U Output parameter.

U is COMPLEX

U is an array, dimension (LDU,3*NSHFTS-3) .

LDU Input parameter.

LDU is INTEGER

LDU is the leading dimension of U just as declared in the in the calling subroutine. LDU.GE.3*NSHFTS-3.

NH Input parameter.

NH is INTEGER

NH is the number of columns in array WH available for workspace. NH.GE.1.

WH Output parameter.

WH is COMPLEX

WH is an array, dimension (LDWH, NH) .

LDWH Input parameter.

LDWH is INTEGER

Leading dimension of WH just as declared in the calling procedure. LDWH.GE.3*NSHFTS-3.

NV Input parameter.

NV is INTEGER

NV is the number of rows in WV agailable for workspace. NV.GE.1.

WV Output parameter.

WV is COMPLEX

WV is an array, dimension (LDWV,3*NSHFTS-3) .

LDWV Input parameter.

LDWV is INTEGER

LDWV is the leading dimension of WV as declared in the in the calling subroutine. LDWV.GE.NV.

Related Information

For this routine in other precisions, please see *dlaqr5*, *slaqr5* and *zlaqr5*.

4.17.110 claqsb

claqsb equilibrates a symmetric band matrix A using the scaling factors in the vector S.

Syntax

Fortran specification:

```
use armpl_library

subroutine claqsb(UPLO, N, KD, AB, LDAB, S, SCOND, AMAX, EQUED)
```

C specification:

```
#include "armpl.h"

void claqsb_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
             armpl_singlecomplex_t *ab, const armpl_int_t *ldab,
             const float *s, const float *scond, const float *amax,
             char *equed, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored. = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of super-diagonals of the matrix A if UPLO = 'U', or the number of sub-diagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, if INFO = 0, the triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H of the band matrix A, in the same storage format as A.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

S Input parameter.

S is REAL

S is an array, dimension (N). The scale factors for A.

SCOND Input parameter.

SCOND is REAL

Ratio of the smallest S(i) to the largest S(i).

AMAX Input parameter.

AMAX is REAL

Absolute value of largest matrix entry.

EQUED Output parameter.

EQUED is CHARACTER*1

Specifies whether or not equilibration was done. = 'N': No equilibration. = 'Y': Equilibration was done, i.e., A has been replaced by diag(S) * A * diag(S).

Related Information

For this routine in other precisions, please see *dlaqsb*, *slaqsb* and *zlaqsb*.

4.17.111 claqsp

claqsp equilibrates a symmetric matrix A using the scaling factors in the vector S.

Syntax

Fortran specification:

```
use armpl_library

subroutine claqsp(UPLO, N, AP, S, SCOND, AMAX, EQUED)
```

C specification:

```
#include "armpl.h"

void claqsp_(const char *uplo, const armpl_int_t *n,
             armpl_singlecomplex_t *ap, const float *s, const float *scond,
             const float *amax, char *equed, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored. = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

On exit, the equilibrated matrix: diag(S) * A * diag(S), in the same storage format as A.

S Input parameter.

S is REAL

S is an array, dimension (N). The scale factors for A.

SCOND Input parameter.

SCOND is REAL

Ratio of the smallest S(i) to the largest S(i).

AMAX Input parameter.

AMAX is REAL

Absolute value of largest matrix entry.

EQUED Output parameter.

EQUED is CHARACTER*1

Specifies whether or not equilibration was done. = 'N': No equilibration. = 'Y': Equilibration was done, i.e., A has been replaced by diag(S) * A * diag(S).

Related Information

For this routine in other precisions, please see *dlaqsp*, *slaqsp* and *zlaqsp*.

4.17.112 claqsy

claqsy equilibrates a symmetric matrix A using the scaling factors in the vector S.

Syntax

Fortran specification:

```
use armpl_library

subroutine claqsy(UPLO, N, A, LDA, S, SCOND, AMAX, EQUED)
```

C specification:

```
#include "armpl.h"

void claqsy_(const char *uplo, const armpl_int_t *n, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, const float *s, const float *scond,
             const float *amax, char *equed, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored. = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading n by n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n by n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if EQUED = 'Y', the equilibrated matrix: diag(S) * A * diag(S).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(N,1).

S Input parameter.

S is REAL

S is an array, dimension (N). The scale factors for A.

SCOND Input parameter.

SCOND is REAL

Ratio of the smallest S(i) to the largest S(i).

AMAX Input parameter.

AMAX is REAL

Absolute value of largest matrix entry.

EQUED Output parameter.

EQUED is CHARACTER*1

Specifies whether or not equilibration was done. = 'N': No equilibration. = 'Y': Equilibration was done, i.e., A has been replaced by diag(S) * A * diag(S).

Related Information

For this routine in other precisions, please see *dlaqsy*, *slaqsy* and *zlaqsy*.

4.17.113 clar1v

clar1v computes the (scaled) r-th column of the inverse of the sumbmatrix in rows B1 through BN of the tridiagonal matrix L D L^T - sigma I. When sigma is close to an eigenvalue, the computed vector is an accurate eigenvector. Usually, r corresponds to the index where the eigenvector is largest in magnitude. The following steps accomplish this computation : (a) Stationary qd transform, L D L^T - sigma I = L(+) D(+) $L(+)^T$, (b) Progressive qd transform, L D L^T - sigma I = U(-) D(-) $U(-)^T$, (c) Computation of the diagonal elements of the inverse of

```
L D L**T - sigma I by combining the above transforms, and choosing
r as the index where the diagonal of the inverse is (one of the)
largest in magnitude.
```

(d) Computation of the (scaled) r-th column of the inverse using the

```
twisted factorization obtained by combining the top part of the
the stationary and the bottom part of the progressive transform.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine clar1v(N, B1, BN, LAMBDA, D, L, LD, LLD, PIVMIN, GAPTOL, Z, WANTNC,
                  NEGCNT, ZTZ, MINGMA, R, ISUPPZ, NRMINV, RESID, RQCORR,
                  WORK)
```

C specification:

```
#include "armpl.h"

void clar1v_(const armpl_int_t *n, const armpl_int_t *b1,
             const armpl_int_t *bn, const float *lambda, const float *d,
             const float *l, const float *ld, const float *lld,
             const float *pivmin, const float *gaptol,
             armpl_singlecomplex_t *z, const armpl_int_t *wantnc,
             armpl_int_t *negcnt, float *ztz, float *mingma, armpl_int_t *r,
             armpl_int_t *isuppz, float *nrminv, float *resid, float *rqcorr,
             float *work);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix L D L^T .

B1 Input parameter.

B1 is INTEGER

First index of the submatrix of L D L^T .

BN Input parameter.

BN is INTEGER

Last index of the submatrix of L D L^T .

LAMBDA Input parameter.

LAMBDA is REAL

The shift. In order to compute an accurate eigenvector, LAMBDA should be a good approximation to an eigenvalue of L D L^T .

L Input parameter.

L is REAL

L is an array, dimension (N-1). The (n-1) subdiagonal elements of the unit bidiagonal matrix L, in elements 1 to N-1.

D Input parameter.

D is REAL

D is an array, dimension (N). The n diagonal elements of the diagonal matrix D.

LD Input parameter.

LD is REAL

LD is an array, dimension (N-1). The n-1 elements L(i)*D(i).

LLD Input parameter.

LLD is REAL

LLD is an array, dimension (N-1). The n-1 elements L(i)*L(i)*D(i).

PIVMIN Input parameter.

PIVMIN is REAL

The minimum pivot in the Sturm sequence.

GAPTOL Input parameter.

GAPTOL is REAL

Tolerance that indicates when eigenvector entries are negligible w.r.t. their contribution to the residual.

Z Input and output parameter.

Z is COMPLEX

Z is an array, dimension (N). On input, all entries of Z must be set to 0. On output, Z contains the (scaled) r-th column of the inverse. The scaling is such that Z(R) equals 1.

WANTNC Input parameter.

WANTNC is LOGICAL

Specifies whether NEGCNT has to be computed.

NEGCNT Output parameter.

NEGCNT is INTEGER

If WANTNC is .TRUE. then NEGCNT = the number of pivots < pivmin in the matrix factorization L D L^T , and NEGCNT = -1 otherwise.

ZTZ Output parameter.

ZTZ is REAL

The square of the 2-norm of Z.

MINGMA Output parameter.

MINGMA is REAL

The reciprocal of the largest (in magnitude) diagonal element of the inverse of L D L^T - sigma I.

R Input and output parameter.

R is INTEGER

The twist index for the twisted factorization used to compute Z. On input, 0 <= R <= N. If R is input as 0, R is set to the index where (L D L^T - sigma I)^{-1} is largest in magnitude. If 1 <= R <= N, R is unchanged. On output, R contains the twist index used to compute Z. Ideally, R designates the position of the maximum entry in the eigenvector.

ISUPPZ Output parameter.

ISUPPZ is INTEGER array, dimension (2)

The support of the vector in Z, i.e., the vector Z is nonzero only in elements ISUPPZ(1) through ISUPPZ(2).

NRMINV Output parameter.

NRMINV is REAL

NRMINV = 1/SQRT(ZTZ)

RESID Output parameter.

RESID is REAL

The residual of the FP vector. RESID = ABS(MINGMA)/SQRT(ZTZ)

RQCORR Output parameter.

RQCORR is REAL

The Rayleigh Quotient correction to LAMBDA. RQCORR = MINGMA*TMP

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (4*N) .

Related Information

For this routine in other precisions, please see *dlar1v*, *slar1v* and *zlar1v*.

4.17.114 clar2v

clar2v applies a vector of complex plane rotations with real cosines from both sides to a sequence of 2-by-2 complex Hermitian matrices, defined by the elements of the vectors x, y and z. For i = 1,2,...,n

```
(       x(i)  z(i) ) :=
( conjg(z(i)) y(i) )

  (  c(i) conjg(s(i)) ) (       x(i)  z(i) ) ( c(i) -conjg(s(i)) )
  ( -s(i)       c(i)  ) ( conjg(z(i)) y(i) ) ( s(i)        c(i)  )
```

Syntax

Fortran specification:

```
use armpl_library

subroutine clar2v(N, X, Y, Z, INCX, C, S, INCC)
```

C specification:

```
#include "armpl.h"

void clar2v_(const armpl_int_t *n, armpl_singlecomplex_t *x,
             armpl_singlecomplex_t *y, armpl_singlecomplex_t *z,
             const armpl_int_t *incx, const float *c,
             const armpl_singlecomplex_t *s, const armpl_int_t *incc);
```

Parameters

N Input parameter.

N is INTEGER

The number of plane rotations to be applied.

X Input and output parameter.

X is COMPLEX

X is an array, dimension (1+(N-1)*INCX). The vector x; the elements of x are assumed to be real.

Y Input and output parameter.

Y is COMPLEX

Y is an array, dimension (1+(N-1)*INCX). The vector y; the elements of y are assumed to be real.

Z Input and output parameter.

Z is COMPLEX

Z is an array, dimension (1+(N-1)*INCX). The vector z.

INCX Input parameter.

INCX is INTEGER

The increment between elements of X, Y and Z. INCX > 0.

C Input parameter.

C is REAL

C is an array, dimension (1+(N-1)*INCC). The cosines of the plane rotations.

S Input parameter.

S is COMPLEX

S is an array, dimension (1+(N-1)*INCC). The sines of the plane rotations.

INCC Input parameter.

INCC is INTEGER

The increment between elements of C and S. INCC > 0.

Related Information

For this routine in other precisions, please see *dlar2v*, *slar2v* and *zlar2v*.

4.17.115 clarcm

clarcm performs a very simple matrix-matrix multiplication:

```
C := A * B,
```

where A is M by M and real; B is M by N and complex; C is M by N and complex.

Syntax

Fortran specification:

```
use armpl_library

subroutine clarcm(M, N, A, LDA, B, LDB, C, LDC, RWORK)
```

C specification:

```
#include "armpl.h"

void clarcm_(const armpl_int_t *m, const armpl_int_t *n, const float *a,
             const armpl_int_t *lda, const armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, armpl_singlecomplex_t *c,
             const armpl_int_t *ldc, float *rwork);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A and of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns and rows of the matrix B and the number of columns of the matrix C. N >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, M). On entry, A contains the M by M matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >=max(1, M).

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, N). On entry, B contains the M by N matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >=max(1, M).

C Output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). On exit, C contains the M by N matrix C.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >=max(1, M).

RWORK Output parameter.

RWORK is REAL

RWORK is an array, dimension (2*M*N) .

Related Information

For this routine in other precisions, please see *zlarcm*. It also exists with a native C interface as *LAPACKE_clarcm*.

4.17.116 clarf

clarf applies a complex elementary reflector H to a complex M-by-N matrix C, from either the left or the right. H is represented in the form

```
H = I - tau * v * v**H
```

where tau is a complex scalar and v is a complex vector.

If tau = 0, then H is taken to be the unit matrix.

To apply H^H (the conjugate transpose of H), supply conjg(tau) instead tau.

Syntax

Fortran specification:

```
use armpl_library

subroutine clarf(SIDE, M, N, V, INCV, TAU, C, LDC, WORK)
```

C specification:

```
#include "armpl.h"

void clarf_(const char *side, const armpl_int_t *m, const armpl_int_t *n,
            const armpl_singlecomplex_t *v, const armpl_int_t *incv,
            const armpl_singlecomplex_t *tau, armpl_singlecomplex_t *c,
            const armpl_int_t *ldc, armpl_singlecomplex_t *work, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': form H * C = 'R': form C * H

M Input parameter.

M is INTEGER

The number of rows of the matrix C.

N Input parameter.

N is INTEGER

The number of columns of the matrix C.

V Input parameter.

V is COMPLEX

V is an array, dimension. (1 + (M-1)*abs(INCV)) if SIDE = 'L' or (1 + (N-1)*abs(INCV)) if SIDE = 'R' The vector v in the representation of H. V is not used if TAU = 0.

INCV Input parameter.

INCV is INTEGER

The increment between elements of v. INCV <> 0.

TAU Input parameter.

TAU is COMPLEX

The value tau in the representation of H.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by the matrix H * C if SIDE = 'L', or C * H if SIDE = 'R'.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension. (N) if SIDE = 'L' or (M) if SIDE = 'R'

Related Information

For this routine in other precisions, please see *dlarf*, *slarf* and *zlarf*.

4.17.117 clarfb

clarfb applies a complex block reflector H or its transpose H^H to a complex M-by-N matrix C, from either the left or the right.

Syntax

Fortran specification:

```
use armpl_library

subroutine clarfb(SIDE, TRANS, DIRECT, STOREV, M, N, K, V, LDV, T, LDT, C,
                  LDC, WORK, LDWORK)
```

C specification:

```
#include "armpl.h"

void clarfb_(const char *side, const char *trans, const char *direct,
             const char *storev, const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *k, const armpl_singlecomplex_t *v,
             const armpl_int_t *ldv, const armpl_singlecomplex_t *t,
             const armpl_int_t *ldt, armpl_singlecomplex_t *c,
```

(continues on next page)

(continued from previous page)

```
            const armpl_int_t *ldc, armpl_singlecomplex_t *work,
            const armpl_int_t *ldwork, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply H or H^H from the Left = 'R': apply H or H^H from the Right

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply H (No transpose) = 'C': apply H^H (Conjugate transpose)

DIRECT Input parameter.

DIRECT is CHARACTER*1

Indicates how H is formed from a product of elementary reflectors = 'F': H = H(1) H(2) . . . H(k) (Forward) = 'B': H = H(k) . . . H(2) H(1) (Backward)

STOREV Input parameter.

STOREV is CHARACTER*1

Indicates how the vectors which define the elementary reflectors are stored: = 'C': Columnwise = 'R': Rowwise

M Input parameter.

M is INTEGER

The number of rows of the matrix C.

N Input parameter.

N is INTEGER

The number of columns of the matrix C.

K Input parameter.

K is INTEGER

The order of the matrix T (= the number of elementary reflectors whose product defines the block reflector).

V Input parameter.

V is COMPLEX

V is an array, dimension. (LDV, K) if STOREV = 'C' (LDV, M) if STOREV = 'R' and SIDE = 'L' (LDV, N) if STOREV = 'R' and SIDE = 'R' The matrix V. See Further Details.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. If STOREV = 'C' and SIDE = 'L', LDV >= max(1, M); if STOREV = 'C' and SIDE = 'R', LDV >= max(1, N); if STOREV = 'R', LDV >= K.

T Input parameter.

T is COMPLEX

T is an array, dimension (LDT, K). The triangular K-by-K matrix T in the representation of the block reflector.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= K.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by H* C or H^H * C or C*H or C*H^H .

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (LDWORK, K) .

LDWORK Input parameter.

LDWORK is INTEGER

The leading dimension of the array WORK. If SIDE = 'L', LDWORK >= max(1, N); if SIDE = 'R', LDWORK >= max(1, M).

Related Information

For this routine in other precisions, please see *dlarfb*, *slarfb* and *zlarfb*. It also exists with a native C interface as *LAPACKE_clarfb*.

4.17.118 clarfg

clarfg generates a complex elementary reflector H of order n, such that

```
H**H * ( alpha ) = ( beta ),   H**H * H = I.
       (   x   )   (   0  )
```

where alpha and beta are scalars, with beta real, and x is an (n-1)-element complex vector. H is represented in the form

```
H = I - tau * ( 1 ) * ( 1 v**H ) ,
              ( v )
```

where tau is a complex scalar and v is a complex (n-1)-element vector. Note that H is not hermitian.

If the elements of x are all zero and alpha is real, then tau = 0 and H is taken to be the unit matrix.

Otherwise 1 <= real(tau) <= 2 and abs(tau-1) <= 1 .

Syntax

Fortran specification:

```
use armpl_library

subroutine clarfg(N, ALPHA, X, INCX, TAU)
```

C specification:

```
#include "armpl.h"

void clarfg_(const armpl_int_t *n, armpl_singlecomplex_t *alpha,
             armpl_singlecomplex_t *x, const armpl_int_t *incx,
             armpl_singlecomplex_t *tau);
```

Parameters

N Input parameter.

N is INTEGER

The order of the elementary reflector.

ALPHA Input and output parameter.

ALPHA is COMPLEX

On entry, the value alpha. On exit, it is overwritten with the value beta.

X Input and output parameter.

X is COMPLEX

X is an array, dimension. (1+(N-2)*abs(INCX)) On entry, the vector x. On exit, it is overwritten with the vector v.

INCX Input parameter.

INCX is INTEGER

The increment between elements of X. INCX > 0.

TAU Output parameter.

TAU is COMPLEX

The value tau.

Related Information

For this routine in other precisions, please see *dlarfg*, *slarfg* and *zlarfg*. It also exists with a native C interface as *LAPACKE_clarfg*.

4.17.119 clarfgp

clarfgp generates a complex elementary reflector H of order n, such that

```
H**H * ( alpha ) = ( beta ),   H**H * H = I.
       (   x   )   (   0  )
```

where alpha and beta are scalars, beta is real and non-negative, and x is an (n-1)-element complex vector. H is represented in the form

```
H = I - tau * ( 1 ) * ( 1 v**H ) ,
              ( v )
```

where tau is a complex scalar and v is a complex (n-1)-element vector. Note that H is not hermitian.

If the elements of x are all zero and alpha is real, then tau = 0 and H is taken to be the unit matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine clarfgp(N, ALPHA, X, INCX, TAU)
```

C specification:

```
#include "armpl.h"

void clarfgp_(const armpl_int_t *n, armpl_singlecomplex_t *alpha,
              armpl_singlecomplex_t *x, const armpl_int_t *incx,
              armpl_singlecomplex_t *tau);
```

Parameters

N Input parameter.

N is INTEGER

The order of the elementary reflector.

ALPHA Input and output parameter.

ALPHA is COMPLEX

On entry, the value alpha. On exit, it is overwritten with the value beta.

X Input and output parameter.

X is COMPLEX

X is an array, dimension. (1+(N-2)*abs(INCX)) On entry, the vector x. On exit, it is overwritten with the vector v.

INCX Input parameter.

INCX is INTEGER

The increment between elements of X. INCX > 0.

TAU Output parameter.

TAU is COMPLEX

The value tau.

Related Information

For this routine in other precisions, please see *dlarfgp*, *slarfgp* and *zlarfgp*.

4.17.120 clarft

clarft forms the triangular factor T of a complex block reflector H of order n, which is defined as a product of k elementary reflectors.

If DIRECT = 'F', H = H(1) H(2) . . . H(k) and T is upper triangular;

If DIRECT = 'B', H = H(k) . . . H(2) H(1) and T is lower triangular.

If STOREV = 'C', the vector which defines the elementary reflector H(i) is stored in the i-th column of the array V, and

```
H  =  I - V * T * V**H
```

If STOREV = 'R', the vector which defines the elementary reflector H(i) is stored in the i-th row of the array V, and

```
H  =  I - V**H * T * V
```

Syntax

Fortran specification:

```
use armpl_library

subroutine clarft(DIRECT, STOREV, N, K, V, LDV, TAU, T, LDT)
```

C specification:

```
#include "armpl.h"

void clarft_(const char *direct, const char *storev, const armpl_int_t *n,
             const armpl_int_t *k, const armpl_singlecomplex_t *v,
             const armpl_int_t *ldv, const armpl_singlecomplex_t *tau,
             armpl_singlecomplex_t *t, const armpl_int_t *ldt, ... );
```

Parameters

DIRECT Input parameter.

DIRECT is CHARACTER*1

Specifies the order in which the elementary reflectors are multiplied to form the block reflector: = 'F': H = H(1) H(2) ... H(k) (Forward) = 'B': H = H(k) ... H(2) H(1) (Backward)

STOREV Input parameter.

STOREV is CHARACTER*1

Specifies how the vectors which define the elementary reflectors are stored (see also Further Details): = 'C': columnwise = 'R': rowwise

N Input parameter.

N is INTEGER

The order of the block reflector H. N >= 0.

K Input parameter.

K is INTEGER

The order of the triangular factor T (= the number of elementary reflectors). K >= 1.

V Input parameter.

V is COMPLEX

V is an array, dimension. (LDV, K) if STOREV = 'C' (LDV, N) if STOREV = 'R' The matrix V. See further details.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. If STOREV = 'C', LDV >= max(1, N); if STOREV = 'R', LDV >= K.

TAU Input parameter.

TAU is COMPLEX

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i).

T Output parameter.

T is COMPLEX

T is an array, dimension (LDT, K). The k by k triangular factor T of the block reflector. If DIRECT = 'F', T is upper triangular; if DIRECT = 'B', T is lower triangular. The rest of the array is not used.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= K.

Related Information

For this routine in other precisions, please see *dlarft*, *slarft* and *zlarft*. It also exists with a native C interface as *LAPACKE_clarft*.

4.17.121 clarfx

clarfx applies a complex elementary reflector H to a complex m by n matrix C, from either the left or the right. H is represented in the form

```
H = I - tau * v * v**H
```

where tau is a complex scalar and v is a complex vector.

If tau = 0, then H is taken to be the unit matrix

This version uses inline code if H has order < 11.

Syntax

Fortran specification:

```
use armpl_library

subroutine clarfx(SIDE, M, N, V, TAU, C, LDC, WORK)
```

C specification:

```
#include "armpl.h"

void clarfx_(const char *side, const armpl_int_t *m, const armpl_int_t *n,
             const armpl_singlecomplex_t *v, const armpl_singlecomplex_t *tau,
             armpl_singlecomplex_t *c, const armpl_int_t *ldc,
             armpl_singlecomplex_t *work, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': form H * C = 'R': form C * H

M Input parameter.

M is INTEGER

The number of rows of the matrix C.

N Input parameter.

N is INTEGER

The number of columns of the matrix C.

V Input parameter.

V is COMPLEX

V is an array, dimension (M) if SIDE = 'L'. or (N) if SIDE = 'R' The vector v in the representation of H.

TAU Input parameter.

TAU is COMPLEX

The value tau in the representation of H.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). On entry, the m by n matrix C. On exit, C is overwritten by the matrix H * C if SIDE = 'L', or C * H if SIDE = 'R'.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDA >= max(1, M).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (N) if SIDE = 'L'. or (M) if SIDE = 'R' WORK is not referenced if H has order < 11.

Related Information

For this routine in other precisions, please see *dlarfx*, *slarfx* and *zlarfx*. It also exists with a native C interface as *LAPACKE_clarfx*.

4.17.122 clarfy

clarfy applies an elementary reflector, or Householder matrix, H, to an n x n Hermitian matrix C, from both the left and the right.

H is represented in the form

```
H = I - tau * v * v'
```

where tau is a scalar and v is a vector.

If tau is zero, then H is taken to be the unit matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine clarfy(UPLO, N, V, INCV, TAU, C, LDC, WORK)
```

C specification:

```
#include "armpl.h"

void clarfy_(const char *uplo, const armpl_int_t *n,
             const armpl_singlecomplex_t *v, const armpl_int_t *incv,
             const armpl_singlecomplex_t *tau, armpl_singlecomplex_t *c,
             const armpl_int_t *ldc, armpl_singlecomplex_t *work, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the Hermitian matrix C is stored. = 'U': Upper triangle = 'L': Lower triangle

N Input parameter.

N is INTEGER

The number of rows and columns of the matrix C. N >= 0.

V Input parameter.

V is COMPLEX

V is an array, dimension. (1 + (N-1)*abs(INCV)) The vector v as described above.

INCV Input parameter.

INCV is INTEGER

The increment between successive elements of v. INCV must not be zero.

TAU Input parameter.

TAU is COMPLEX

The value tau as described above.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). On entry, the matrix C. On exit, C is overwritten by H * C * H'.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, N).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (N) .

Related Information

For this routine in other precisions, please see *dlarfy*, *slarfy* and *zlarfy*.

4.17.123 clargv

clargv generates a vector of complex plane rotations with real cosines, determined by elements of the complex vectors x and y. For i = 1,2,...,n

```
(        c(i)   s(i) ) ( x(i) ) = ( r(i) )
( -conjg(s(i))  c(i) ) ( y(i) ) = (   0  )

where c(i)**2 + ABS(s(i))**2 = 1
```

The following conventions are used (these are the same as in CLARTG, but differ from the BLAS1 routine CROTG):

```
 If y(i)=0, then c(i)=1 and s(i)=0.
If x(i)=0, then c(i)=0 and s(i) is chosen so that r(i) is real.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine clargv(N, X, INCX, Y, INCY, C, INCC)
```

C specification:

```
#include "armpl.h"

void clargv_(const armpl_int_t *n, armpl_singlecomplex_t *x,
             const armpl_int_t *incx, armpl_singlecomplex_t *y,
             const armpl_int_t *incy, float *c, const armpl_int_t *incc);
```

Parameters

N Input parameter.

N is INTEGER

The number of plane rotations to be generated.

X Input and output parameter.

X is COMPLEX

X is an array, dimension (1+(N-1)*INCX). On entry, the vector x. On exit, x(i) is overwritten by r(i), for i = 1,...,n.

INCX Input parameter.

INCX is INTEGER

The increment between elements of X. INCX > 0.

Y Input and output parameter.

Y is COMPLEX

Y is an array, dimension (1+(N-1)*INCY). On entry, the vector y. On exit, the sines of the plane rotations.

INCY Input parameter.

INCY is INTEGER

The increment between elements of Y. INCY > 0.

C Output parameter.

C is REAL

C is an array, dimension (1+(N-1)*INCC). The cosines of the plane rotations.

INCC Input parameter.

INCC is INTEGER

The increment between elements of C. INCC > 0.

Related Information

For this routine in other precisions, please see *dlargv*, *slargv* and *zlargv*.

4.17.124 clarnv

clarnv returns a vector of n random complex numbers from a uniform or normal distribution.

Syntax

Fortran specification:

```
use armpl_library

subroutine clarnv(IDIST, ISEED, N, X)
```

C specification:

```
#include "armpl.h"

void clarnv_(const armpl_int_t *idist, armpl_int_t *iseed,
             const armpl_int_t *n, armpl_singlecomplex_t *x);
```

Parameters

IDIST Input parameter.

IDIST is INTEGER

Specifies the distribution of the random numbers: = 1: real and imaginary parts each uniform (0,1) = 2: real and imaginary parts each uniform (-1,1) = 3: real and imaginary parts each normal (0,1) = 4: uniformly distributed on the disc abs(z) < 1 = 5: uniformly distributed on the circle abs(z) = 1

ISEED Input and output parameter.

ISEED is INTEGER array, dimension (4)

On entry, the seed of the random number generator; the array elements must be between 0 and 4095, and ISEED(4) must be odd. On exit, the seed is updated.

N Input parameter.

N is INTEGER

The number of random numbers to be generated.

X Output parameter.

X is COMPLEX

X is an array, dimension (N). The generated random numbers.

Related Information

For this routine in other precisions, please see *dlarnv*, *slarnv* and *zlarnv*. It also exists with a native C interface as *LAPACKE_clarnv*.

4.17.125 clarrv

clarrv computes the eigenvectors of the tridiagonal matrix T = L D L^T given L, D and APPROXIMATIONS to the eigenvalues of L D L^T . The input eigenvalues should have been computed by SLARRE.

Syntax

Fortran specification:

```
use armpl_library

subroutine clarrv(N, VL, VU, D, L, PIVMIN, ISPLIT, M, DOL, DOU, MINRGP, RTOL1,
                  RTOL2, W, WERR, WGAP, IBLOCK, INDEXW, GERS, Z, LDZ, ISUPPZ,
                  WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void clarrv_(const armpl_int_t *n, const float *vl, const float *vu, float *d,
             float *l, float *pivmin, const armpl_int_t *isplit,
             const armpl_int_t *m, const armpl_int_t *dol,
             const armpl_int_t *dou, const float *minrgp, const float *rtol1,
             const float *rtol2, float *w, float *werr, float *wgap,
             const armpl_int_t *iblock, const armpl_int_t *indexw,
             const float *gers, armpl_singlecomplex_t *z,
             const armpl_int_t *ldz, armpl_int_t *isuppz, float *work,
             armpl_int_t *iwork, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix. N >= 0.

VL Input parameter.

VL is REAL

Lower bound of the interval that contains the desired eigenvalues. VL < VU. Needed to compute gaps on the left or right end of the extremal eigenvalues in the desired RANGE.

VU Input parameter.

VU is REAL

Upper bound of the interval that contains the desired eigenvalues. VL < VU. Needed to compute gaps on the left or right end of the extremal eigenvalues in the desired RANGE.

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry, the N diagonal elements of the diagonal matrix D. On exit, D may be overwritten.

L Input and output parameter.

L is REAL

L is an array, dimension (N). On entry, the (N-1) subdiagonal elements of the unit bidiagonal matrix L are in elements 1 to N-1 of L (if the matrix is not split.) At the end of each block is stored the corresponding shift as given by SLARRE. On exit, L is overwritten.

PIVMIN Input parameter.

PIVMIN is REAL

The minimum pivot allowed in the Sturm sequence.

ISPLIT Input parameter.

ISPLIT is INTEGER array, dimension (N)

The splitting points, at which T breaks up into blocks. The first block consists of rows/columns 1 to ISPLIT(1), the second of rows/columns ISPLIT(1)+1 through ISPLIT(2), etc.

M Input parameter.

M is INTEGER

The total number of input eigenvalues. 0 <= M <= N.

DOL Input parameter.

DOL is INTEGER

DOU Input parameter.

DOU is INTEGER

If the user wants to compute only selected eigenvectors from all the eigenvalues supplied, he can specify an index range DOL:DOU. Or else the setting DOL=1, DOU=M should be applied. Note that DOL and DOU refer to the order in which the eigenvalues are stored in W. If the user wants to compute only selected eigenpairs, then the columns DOL-1 to DOU+1 of the eigenvector space Z contain the computed eigenvectors. All other columns of Z are set to zero.

MINRGP Input parameter.

MINRGP is REAL

RTOL1 Input parameter.

RTOL1 is REAL

RTOL2 Input parameter.

RTOL2 is REAL

Parameters for bisection. An interval [LEFT,RIGHT] has converged if RIGHT-LEFT.LT.MAX(RTOL1*GAP, RTOL2*MAX(|LEFT|,|RIGHT|))

W Input and output parameter.

W is REAL

W is an array, dimension (N). The first M elements of W contain the APPROXIMATE eigenvalues for which eigenvectors are to be computed. The eigenvalues should be grouped by split-off block and ordered from smallest to largest within the block (The output array W from SLARRE is expected here). Furthermore, they are with respect to the shift of the corresponding root representation for their block. On exit, W holds the eigenvalues of the UNshifted matrix.

WERR Input and output parameter.

WERR is REAL

WERR is an array, dimension (N). The first M elements contain the semiwidth of the uncertainty interval of the corresponding eigenvalue in W

WGAP Input and output parameter.

WGAP is REAL

WGAP is an array, dimension (N). The separation from the right neighbor eigenvalue in W.

IBLOCK Input parameter.

IBLOCK is INTEGER array, dimension (N)

The indices of the blocks (submatrices) associated with the corresponding eigenvalues in W; IBLOCK(i)=1 if eigenvalue W(i) belongs to the first block from the top, =2 if W(i) belongs to the second block, etc.

INDEXW Input parameter.

INDEXW is INTEGER array, dimension (N)

The indices of the eigenvalues within each block (submatrix); for example, INDEXW(i)= 10 and IBLOCK(i)=2 imply that the i-th eigenvalue W(i) is the 10-th eigenvalue in the second block.

GERS Input parameter.

GERS is REAL

GERS is an array, dimension (2*N). The N Gerschgorin intervals (the i-th Gerschgorin interval is (GERS(2*i-1), GERS(2*i)). The Gerschgorin intervals should be computed from the original UNshifted matrix.

Z Output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, max(1, M)). If INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix T corresponding to the input eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). Note: the user must ensure that at least max(1, M) columns are supplied in the array Z.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

ISUPPZ Output parameter.

ISUPPZ is INTEGER array, dimension (2*max(1, M))

The support of the eigenvectors in Z, i.e., the indices indicating the nonzero elements in Z. The I-th eigenvector is nonzero only in elements ISUPPZ(2*I-1) through ISUPPZ(2*I).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (12*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (7*N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit

> 0: A problem occurred in CLARRV. < 0: One of the called subroutines signaled an internal problem. Needs inspection of the corresponding parameter IINFO for further information.

=-1: Problem in SLARRB when refining a child's eigenvalues. =-2: Problem in SLARRF when computing the RRR of a child. When a child is inside a tight cluster, it can be difficult to find an RRR. A partial remedy from the user's point of view is to make the parameter MINRGP smaller and recompile. However, as the orthogonality of the computed vectors is proportional to 1/MINRGP, the user should be aware that he might be trading in precision when he decreases MINRGP. =-3: Problem in SLARRB when refining a single eigenvalue after the Rayleigh correction was rejected. = 5: The Rayleigh Quotient Iteration failed to converge to full accuracy in MAXITR steps.

Related Information

For this routine in other precisions, please see *dlarrv*, *slarrv* and *zlarrv*.

4.17.126 clarscl2

CLARSCL2 performs a reciprocal diagonal scaling on an vector:

```
x <-- inv(D) * x
```

where the REAL diagonal matrix D is stored as a vector.

Eventually to be replaced by BLAS_cge_diag_scale in the new BLAS standard.

Syntax

Fortran specification:

```
use armpl_library

subroutine clarscl2(M, N, D, X, LDX)
```

C specification:

```
#include "armpl.h"

void clarscl2_(const armpl_int_t *m, const armpl_int_t *n, const float *d,
               armpl_singlecomplex_t *x, const armpl_int_t *ldx);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of D and X. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of X. N >= 0.

D Input parameter.

D is REAL

D is an array, length M. Diagonal matrix D, stored as a vector of length M.

X Input and output parameter.

X is COMPLEX

X is an array, dimension (LDX, N). On entry, the vector X to be scaled by D. On exit, the scaled vector.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the vector X. LDX >= M.

Related Information

For this routine in other precisions, please see *dlarscl2*, *slarscl2* and *zlarscl2*.

4.17.127 clartg

clartg generates a plane rotation so that

```
[  CS  SN  ]     [ F ]     [ R ]
[  __      ]  .  [   ]  =  [   ]   where CS**2 + |SN|**2 = 1.
[ -SN  CS  ]     [ G ]     [ 0 ]
```

This is a faster version of the BLAS1 routine CROTG, except for the following differences:

```
F and G are unchanged on return.
If G=0, then CS=1 and SN=0.
If F=0, then CS=0 and SN is chosen so that R is real.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine clartg(F, G, CS, SN, R)
```

C specification:

```
#include "armpl.h"

void clartg_(const armpl_singlecomplex_t *f, const armpl_singlecomplex_t *g,
             float *cs, armpl_singlecomplex_t *sn, armpl_singlecomplex_t *r);
```

Parameters

F Input parameter.

F is COMPLEX

The first component of vector to be rotated.

G Input parameter.

G is COMPLEX

The second component of vector to be rotated.

CS Output parameter.

CS is REAL

The cosine of the rotation.

SN Output parameter.

SN is COMPLEX

The sine of the rotation.

R Output parameter.

R is COMPLEX

The nonzero component of the rotated vector.

Related Information

For this routine in other precisions, please see *dlartg*, *slartg* and *zlartg*.

4.17.128 clartv

clartv applies a vector of complex plane rotations with real cosines to elements of the complex vectors x and y. For i = 1,2,...,n

```
( x(i) ) := (         c(i)    s(i) ) ( x(i) )
( y(i) )    ( -conjg(s(i))    c(i) ) ( y(i) )
```

Syntax

Fortran specification:

```
use armpl_library

subroutine clartv(N, X, INCX, Y, INCY, C, S, INCC)
```

C specification:

```
#include "armpl.h"

void clartv_(const armpl_int_t *n, armpl_singlecomplex_t *x,
             const armpl_int_t *incx, armpl_singlecomplex_t *y,
             const armpl_int_t *incy, const float *c,
             const armpl_singlecomplex_t *s, const armpl_int_t *incc);
```

Parameters

N Input parameter.

N is INTEGER

The number of plane rotations to be applied.

X Input and output parameter.

X is COMPLEX

X is an array, dimension (1+(N-1)*INCX). The vector x.

INCX Input parameter.

INCX is INTEGER

The increment between elements of X. INCX > 0.

Y Input and output parameter.

Y is COMPLEX

Y is an array, dimension (1+(N-1)*INCY). The vector y.

INCY Input parameter.

INCY is INTEGER

The increment between elements of Y. INCY > 0.

C Input parameter.

C is REAL

C is an array, dimension (1+(N-1)*INCC). The cosines of the plane rotations.

S Input parameter.

S is COMPLEX

S is an array, dimension (1+(N-1)*INCC). The sines of the plane rotations.

INCC Input parameter.

INCC is INTEGER

The increment between elements of C and S. INCC > 0.

Related Information

For this routine in other precisions, please see *dlartv*, *slartv* and *zlartv*.

4.17.129 clarz

clarz applies a complex elementary reflector H to a complex M-by-N matrix C, from either the left or the right. H is represented in the form

```
H = I - tau * v * v**H
```

where tau is a complex scalar and v is a complex vector.

If tau = 0, then H is taken to be the unit matrix.

To apply H^H (the conjugate transpose of H), supply conjg(tau) instead tau.

H is a product of k elementary reflectors as returned by CTZRZF.

Syntax

Fortran specification:

```
use armpl_library

subroutine clarz(SIDE, M, N, L, V, INCV, TAU, C, LDC, WORK)
```

C specification:

```
#include "armpl.h"

void clarz_(const char *side, const armpl_int_t *m, const armpl_int_t *n,
            const armpl_int_t *l, const armpl_singlecomplex_t *v,
            const armpl_int_t *incv, const armpl_singlecomplex_t *tau,
            armpl_singlecomplex_t *c, const armpl_int_t *ldc,
            armpl_singlecomplex_t *work, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': form H * C = 'R': form C * H

M Input parameter.

M is INTEGER

The number of rows of the matrix C.

N Input parameter.

N is INTEGER

The number of columns of the matrix C.

L Input parameter.

L is INTEGER

The number of entries of the vector V containing the meaningful part of the Householder vectors. If SIDE = 'L', M >= L >= 0, if SIDE = 'R', N >= L >= 0.

V Input parameter.

V is COMPLEX

V is an array, dimension (1+(L-1)*abs(INCV)). The vector v in the representation of H as returned by CTZRZF. V is not used if TAU = 0.

INCV Input parameter.

INCV is INTEGER

The increment between elements of v. INCV <> 0.

TAU Input parameter.

TAU is COMPLEX

The value tau in the representation of H.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by the matrix H * C if SIDE = 'L', or C * H if SIDE = 'R'.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension. (N) if SIDE = 'L' or (M) if SIDE = 'R'

Related Information

For this routine in other precisions, please see *dlarz*, *slarz* and *zlarz*.

4.17.130 clarzb

clarzb applies a complex block reflector H or its transpose H^H to a complex distributed M-by-N C from the left or the right.

Currently, only STOREV = 'R' and DIRECT = 'B' are supported.

Syntax

Fortran specification:

```
use armpl_library

subroutine clarzb(SIDE, TRANS, DIRECT, STOREV, M, N, K, L, V, LDV, T, LDT, C,
                  LDC, WORK, LDWORK)
```

C specification:

```
#include "armpl.h"

void clarzb_(const char *side, const char *trans, const char *direct,
             const char *storev, const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *k, const armpl_int_t *l,
             const armpl_singlecomplex_t *v, const armpl_int_t *ldv,
             const armpl_singlecomplex_t *t, const armpl_int_t *ldt,
             armpl_singlecomplex_t *c, const armpl_int_t *ldc,
             armpl_singlecomplex_t *work, const armpl_int_t *ldwork, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply H or H^H from the Left = 'R': apply H or H^H from the Right

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply H (No transpose) = 'C': apply H^H (Conjugate transpose)

DIRECT Input parameter.

DIRECT is CHARACTER*1

Indicates how H is formed from a product of elementary reflectors = 'F': H = H(1) H(2) . . . H(k) (Forward, not supported yet) = 'B': H = H(k) . . . H(2) H(1) (Backward)

STOREV Input parameter.

STOREV is CHARACTER*1

Indicates how the vectors which define the elementary reflectors are stored: = 'C': Columnwise (not supported yet) = 'R': Rowwise

M Input parameter.

M is INTEGER

The number of rows of the matrix C.

N Input parameter.

N is INTEGER

The number of columns of the matrix C.

K Input parameter.

K is INTEGER

The order of the matrix T (= the number of elementary reflectors whose product defines the block reflector).

L Input parameter.

L is INTEGER

The number of columns of the matrix V containing the meaningful part of the Householder reflectors. If SIDE = 'L', M >= L >= 0, if SIDE = 'R', N >= L >= 0.

V Input parameter.

V is COMPLEX

V is an array, dimension (LDV,NV).. If STOREV = 'C', NV = K; if STOREV = 'R', NV = L.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. If STOREV = 'C', LDV >= L; if STOREV = 'R', LDV >= K.

T Input parameter.

T is COMPLEX

T is an array, dimension (LDT, K). The triangular K-by-K matrix T in the representation of the block reflector.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= K.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by H* C or H^H * C or C*H or C*H^H .

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (LDWORK, K) .

LDWORK Input parameter.

LDWORK is INTEGER

The leading dimension of the array WORK. If SIDE = 'L', LDWORK >= max(1, N); if SIDE = 'R', LDWORK >= max(1, M).

Related Information

For this routine in other precisions, please see *dlarzb*, *slarzb* and *zlarzb*.

4.17.131 clarzt

`clarzt` forms the triangular factor T of a complex block reflector H of order > n, which is defined as a product of k elementary reflectors.

If DIRECT = 'F', H = H(1) H(2) . . . H(k) and T is upper triangular;

If DIRECT = 'B', H = H(k) . . . H(2) H(1) and T is lower triangular.

If STOREV = '`C`', the vector which defines the elementary reflector H(i) is stored in the i-th column of the array V, and

```
H  =  I - V * T * V**H
```

If STOREV = '`R`', the vector which defines the elementary reflector H(i) is stored in the i-th row of the array V, and

```
H  =  I - V**H * T * V
```

Currently, only STOREV = '`R`' and DIRECT = 'B' are supported.

Syntax

Fortran specification:

```
use armpl_library

subroutine clarzt(DIRECT, STOREV, N, K, V, LDV, TAU, T, LDT)
```

C specification:

```
#include "armpl.h"

void clarzt_(const char *direct, const char *storev, const armpl_int_t *n,
             const armpl_int_t *k, armpl_singlecomplex_t *v,
             const armpl_int_t *ldv, const armpl_singlecomplex_t *tau,
             armpl_singlecomplex_t *t, const armpl_int_t *ldt, ... );
```

Parameters

`DIRECT` Input parameter.

`DIRECT` is `CHARACTER*1`

Specifies the order in which the elementary reflectors are multiplied to form the block reflector: = 'F': H = H(1) H(2) . . . H(k) (Forward, not supported yet) = 'B': H = H(k) . . . H(2) H(1) (Backward)

`STOREV` Input parameter.

`STOREV` is `CHARACTER*1`

Specifies how the vectors which define the elementary reflectors are stored (see also Further Details): = '`C`': columnwise (not supported yet) = '`R`': rowwise

`N` Input parameter.

`N` is `INTEGER`

The order of the block reflector H. `N` >= 0.

`K` Input parameter.

`K` is `INTEGER`

The order of the triangular factor `T` (= the number of elementary reflectors). `K` >= 1.

V Input and output parameter.

V is COMPLEX

V is an array, dimension. (LDV, K) if STOREV = 'C' (LDV, N) if STOREV = 'R' The matrix V. See further details.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. If STOREV = 'C', LDV >= max(1, N); if STOREV = 'R', LDV >= K.

TAU Input parameter.

TAU is COMPLEX

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i).

T Output parameter.

T is COMPLEX

T is an array, dimension (LDT, K). The k by k triangular factor T of the block reflector. If DIRECT = 'F', T is upper triangular; if DIRECT = 'B', T is lower triangular. The rest of the array is not used.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= K.

Related Information

For this routine in other precisions, please see *dlarzt*, *slarzt* and *zlarzt*.

4.17.132 clascl

clascl multiplies the M by N complex matrix A by the real scalar CTO/CFROM. This is done without over/underflow as long as the final result CTO*A(I,J)/CFROM does not over/underflow. TYPE specifies that A may be full, upper triangular, lower triangular, upper Hessenberg, or banded.

Syntax

Fortran specification:

```
use armpl_library

subroutine clascl(TYPE, KL, KU, CFROM, CTO, M, N, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void clascl_(const char *type, const armpl_int_t *kl, const armpl_int_t *ku,
             const float *cfrom, const float *cto, const armpl_int_t *m,
             const armpl_int_t *n, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *info, ... );
```

Parameters

TYPE Input parameter.

TYPE is CHARACTER*1

TYPE indices the storage type of the input matrix. = 'G': A is a full matrix. = 'L': A is a lower triangular matrix. = 'U': A is an upper triangular matrix. = 'H': A is an upper Hessenberg matrix. = 'B': A is a symmetric band matrix with lower bandwidth KL and upper bandwidth KU and with the only the lower half stored. = 'Q': A is a symmetric band matrix with lower bandwidth KL and upper bandwidth KU and with the only the upper half stored. = 'Z': A is a band matrix with lower bandwidth KL and upper bandwidth KU. See CGBTRF for storage details.

KL Input parameter.

KL is INTEGER

The lower bandwidth of A. Referenced only if TYPE = 'B', 'Q' or 'Z'.

KU Input parameter.

KU is INTEGER

The upper bandwidth of A. Referenced only if TYPE = 'B', 'Q' or 'Z'.

CFROM Input parameter.

CFROM is REAL

CTO Input parameter.

CTO is REAL

The matrix A is multiplied by CTO/CFROM. A(I,J) is computed without over/underflow if the final result CTO*A(I,J)/CFROM can be represented without over/underflow. CFROM must be nonzero.

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The matrix to be multiplied by CTO/CFROM. See TYPE for the storage type.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If TYPE = 'G', 'L', 'U', 'H', LDA >= max(1, M); TYPE = 'B', LDA >= KL+1; TYPE = 'Q', LDA >= KU+1; TYPE = 'Z', LDA >= 2*KL+KU+1.

INFO Output parameter.

INFO is INTEGER

0 - successful exit <0 - if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *dlascl*, *slascl* and *zlascl*. It also exists with a native C interface as *LAPACKE_clascl*.

4.17.133 clascl2

`clascl2` performs a diagonal scaling on a vector:

```
x <-- D * x
```

where the diagonal REAL matrix D is stored as a vector.

Eventually to be replaced by BLAS_cge_diag_scale in the new BLAS standard.

Syntax

Fortran specification:

```
use armpl_library

subroutine clascl2(M, N, D, X, LDX)
```

C specification:

```
#include "armpl.h"

void clascl2_(const armpl_int_t *m, const armpl_int_t *n, const float *d,
              armpl_singlecomplex_t *x, const armpl_int_t *ldx);
```

Parameters

M Input parameter.

`M` is `INTEGER`

The number of rows of `D` and `X`. `M` >= 0.

N Input parameter.

`N` is `INTEGER`

The number of columns of `X`. `N` >= 0.

D Input parameter.

`D` is `REAL`

`D` is an array, length `M`. Diagonal matrix `D`, stored as a vector of length `M`.

X Input and output parameter.

`X` is `COMPLEX`

`X` is an array, dimension (`LDX`, `N`). On entry, the vector `X` to be scaled by `D`. On exit, the scaled vector.

LDX Input parameter.

`LDX` is `INTEGER`

The leading dimension of the vector `X`. `LDX` >= `M`.

Related Information

For this routine in other precisions, please see *dlascl2*, *slascl2* and *zlascl2*.

4.17.134 claset

`claset` initializes a 2-D array A to BETA on the diagonal and ALPHA on the offdiagonals.

Syntax

Fortran specification:

```
use armpl_library

subroutine claset(UPLO, M, N, ALPHA, BETA, A, LDA)
```

C specification:

```
#include "armpl.h"

void claset_(const char *uplo, const armpl_int_t *m, const armpl_int_t *n,
             const armpl_singlecomplex_t *alpha,
             const armpl_singlecomplex_t *beta, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies the part of the matrix A to be set. = 'U': Upper triangular part is set. The lower triangle is unchanged. = 'L': Lower triangular part is set. The upper triangle is unchanged. Otherwise: All of the matrix A is set.

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of A.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of A.

ALPHA Input parameter.

ALPHA is COMPLEX

All the offdiagonal array elements are set to ALPHA.

BETA Input parameter.

BETA is COMPLEX

All the diagonal array elements are set to BETA.

A Output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the m by n matrix A. On exit, A(i,j) = ALPHA, 1 <= i <= m, 1 <= j <= n, i.ne.j; A(i,i) = BETA , 1 <= i <= min(m,n)

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

Related Information

For this routine in other precisions, please see *dlaset*, *slaset* and *zlaset*. It also exists with a native C interface as *LAPACKE_claset*.

4.17.135 clasr

clasr applies a sequence of real plane rotations to a complex matrix A, from either the left or the right.

When SIDE = 'L', the transformation takes the form

```
A := P*A
```

and when SIDE = 'R', the transformation takes the form

```
A := A*P**T
```

where P is an orthogonal matrix consisting of a sequence of z plane rotations, with z = M when SIDE = 'L' and z = N when SIDE = 'R', and P^T is the transpose of P.

When DIRECT = 'F' (Forward sequence), then

```
P = P(z-1) * ... * P(2) * P(1)
```

and when DIRECT = 'B' (Backward sequence), then

```
P = P(1) * P(2) * ... * P(z-1)
```

where P(k) is a plane rotation matrix defined by the 2-by-2 rotation

```
R(k) = (  c(k)  s(k) )
     = ( -s(k)  c(k) ).
```

When PIVOT = 'V' (Variable pivot), the rotation is performed for the plane (k,k+1), i.e., P(k) has the form

```
P(k) = (  1                                          )
       (       ...                                   )
       (               1                             )
       (                    c(k)  s(k)               )
       (                   -s(k)  c(k)               )
       (                             1               )
       (                                  ...        )
       (                                         1   )
```

where R(k) appears as a rank-2 modification to the identity matrix in rows and columns k and k+1.

When PIVOT = 'T' (Top pivot), the rotation is performed for the plane (1,k+1), so P(k) has the form

```
P(k) = (  c(k)                    s(k)               )
       (         1                                   )
       (              ...                            )
       (                      1                      )
       ( -s(k)                c(k)                   )
       (                             1               )
       (                                  ...        )
       (                                         1   )
```

where R(k) appears in rows and columns 1 and k+1.

Similarly, when PIVOT = 'B' (Bottom pivot), the rotation is performed for the plane (k,z), giving P(k) the form

```
P(k) = ( 1                                          )
       (      ...                                   )
       (              1                             )
       (                    c(k)           s(k)     )
       (                           1                )
       (                                  ...       )
       (                                        1   )
       (                   -s(k)          c(k)     )
```

where R(k) appears in rows and columns k and z. The rotations are performed without ever forming P(k) explicitly.

Syntax

Fortran specification:

```
use armpl_library

subroutine clasr(SIDE, PIVOT, DIRECT, M, N, C, S, A, LDA)
```

C specification:

```
#include "armpl.h"

void clasr_(const char *side, const char *pivot, const char *direct,
            const armpl_int_t *m, const armpl_int_t *n, const float *c,
            const float *s, armpl_singlecomplex_t *a, const armpl_int_t *lda,
            ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

Specifies whether the plane rotation matrix P is applied to A on the left or the right. = 'L': Left, compute A := P*A = 'R': Right, compute A:= A*P^T

PIVOT Input parameter.

PIVOT is CHARACTER*1

Specifies the plane for which P(k) is a plane rotation matrix. = 'V': Variable pivot, the plane (k,k+1) = 'T': Top pivot, the plane (1,k+1) = 'B': Bottom pivot, the plane (k,z)

DIRECT Input parameter.

DIRECT is CHARACTER*1

Specifies whether P is a forward or backward sequence of plane rotations. = 'F': Forward, P = P(z-1)*...*P(2)*P(1) = 'B': Backward, P = P(1)*P(2)*...*P(z-1)

M Input parameter.

M is INTEGER

The number of rows of the matrix A. If m <= 1, an immediate return is effected.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. If n <= 1, an immediate return is effected.

C Input parameter.

C is REAL

C is an array, dimension. (M-1) if SIDE = 'L' (N-1) if SIDE = 'R' The cosines c(k) of the plane rotations.

S Input parameter.

S is REAL

S is an array, dimension. (M-1) if SIDE = 'L' (N-1) if SIDE = 'R' The sines s(k) of the plane rotations. The 2-by-2 plane rotation part of the matrix P(k), R(k), has the form R(k) = (c(k) s(k)) (-s(k) c(k)).

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The M-by-N matrix A. On exit, A is overwritten by P* A if SIDE = 'R' or by A*P^T if SIDE = 'L'.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

Related Information

For this routine in other precisions, please see *dlasr*, *slasr* and *zlasr*.

4.17.136 classq

classq returns the values scl and ssq such that

```
( scl**2 )*ssq = x( 1 )**2 +...+ x( n )**2 + ( scale**2 )*sumsq,
```

where x(i) = abs(X(1 + (i - 1)*INCX)). The value of sumsq is assumed to be at least unity and the value of ssq will then satisfy

```
1.0 .le. ssq .le. ( sumsq + 2*n ).
```

scale is assumed to be non-negative and scl returns the value

```
scl = max( scale, abs( real( x( i ) ) ), abs( aimag( x( i ) ) ) ),
      i
```

scale and sumsq must be supplied in SCALE and SUMSQ respectively. SCALE and SUMSQ are overwritten by scl and ssq respectively.

The routine makes only one pass through the vector X.

Syntax

Fortran specification:

```
use armpl_library

subroutine classq(N, X, INCX, SCALE, SUMSQ)
```

C specification:

```
#include "armpl.h"

void classq_(const armpl_int_t *n, const armpl_singlecomplex_t *x,
             const armpl_int_t *incx, float *scale, float *sumsq);
```

Parameters

N Input parameter.

N is INTEGER

The number of elements to be used from the vector X.

X Input parameter.

X is COMPLEX

X is an array, dimension (N). The vector x as described above. x(i) = X(1 + (i - 1)* INCX), 1 <= i <= n.

INCX Input parameter.

INCX is INTEGER

The increment between successive values of the vector X. INCX > 0.

SCALE Input and output parameter.

SCALE is REAL

On entry, the value scale in the equation above. On exit, SCALE is overwritten with the value scl .

SUMSQ Input and output parameter.

SUMSQ is REAL

On entry, the value sumsq in the equation above. On exit, SUMSQ is overwritten with the value ssq .

Related Information

For this routine in other precisions, please see *dlassq*, *slassq* and *zlassq*. It also exists with a native C interface as *LAPACKE_classq*.

4.17.137 claswlq

claswlq computes a blocked Short-Wide LQ factorization of a M-by-N matrix A, where N >= M: A = L * Q

Syntax

Fortran specification:

```
use armpl_library

subroutine claswlq(M, N, MB, NB, A, LDA, T, LDT, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void claswlq_(const armpl_int_t *m, const armpl_int_t *n,
              const armpl_int_t *mb, const armpl_int_t *nb,
              armpl_singlecomplex_t *a, const armpl_int_t *lda,
              armpl_singlecomplex_t *t, const armpl_int_t *ldt,
```

(continues on next page)

(continued from previous page)

```
                armpl_singlecomplex_t *work, const armpl_int_t *lwork,
                armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= M >= 0.

MB Input parameter.

MB is INTEGER

The row block size to be used in the blocked QR. M >= MB >= 1

NB Input parameter.

NB is INTEGER

The column block size to be used in the blocked QR. NB > M.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and below the diagonal of the array contain the N-by-N lower triangular matrix L; the elements above the diagonal represent Q by the rows of blocked V (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

T Output parameter.

T is COMPLEX

T is an array,. dimension (LDT, N * Number_of_row_blocks) where Number_of_row_blocks = CEIL((N-M)/(NB-M)) The blocked upper triangular block reflectors stored in compact form as a sequence of upper triangular blocks. See Further Details below.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= MB.

WORK Output parameter.

(workspace) COMPLEX

(workspace) is an array, dimension (MAX(1, LWORK)) .

LWORK Input parameter.

The dimension of the array WORK. LWORK >= MB*M.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dlaswlq*, *slaswlq* and *zlaswlq*.

4.17.138 claswp

claswp performs a series of row interchanges on the matrix A. One row interchange is initiated for each of rows K1 through K2 of A.

Syntax

Fortran specification:

```
use armpl_library

subroutine claswp(N, A, LDA, K1, K2, IPIV, INCX)
```

C specification:

```
#include "armpl.h"

void claswp_(const armpl_int_t *n, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, const armpl_int_t *k1,
             const armpl_int_t *k2, const armpl_int_t *ipiv,
             const armpl_int_t *incx);
```

Parameters

N Input parameter.

N is INTEGER

The number of columns of the matrix A.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the matrix of column dimension N to which the row interchanges will be applied. On exit, the permuted matrix.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A.

K1 Input parameter.

K1 is INTEGER

The first element of IPIV for which a row interchange will be done.

K2 Input parameter.

K2 is INTEGER

(K2-K1+1) is the number of elements of IPIV for which a row interchange will be done.

IPIV Input parameter.

IPIV is INTEGER array, dimension (K1+(K2-K1)*abs(INCX))

The vector of pivot indices. Only the elements in positions K1 through K1+(K2-K1)*abs(INCX) of IPIV are accessed. IPIV(K1+(K-K1)*abs(INCX)) = L implies rows K and L are to be interchanged.

INCX Input parameter.

INCX is INTEGER

The increment between successive values of IPIV. If INCX is negative, the pivots are applied in reverse order.

Related Information

For this routine in other precisions, please see *dlaswp*, *slaswp* and *zlaswp*. It also exists with a native C interface as *LAPACKE_claswp*.

4.17.139 clasyf

clasyf computes a partial factorization of a complex symmetric matrix A using the Bunch-Kaufman diagonal pivoting method. The partial factorization has the form:

A = (I U12) (A11 0) (I 0) if UPLO = 'U', or:

```
( 0  U22 ) (  0   D  ) ( U12**T U22**T )
```

A = (L11 0) (D 0) ($L11^T$ $L21^T$) if UPLO = 'L'

```
( L21  I ) ( 0   A22 ) (  0       I    )
```

where the order of D is at most NB. The actual order is returned in the argument KB, and is either NB or NB-1, or N if N <= NB. Note that U^T denotes the transpose of U.

clasyf is an auxiliary routine called by CSYTRF. It uses blocked code (calling Level 3 BLAS) to update the submatrix A11 (if UPLO = 'U') or A22 (if UPLO = 'L').

Syntax

Fortran specification:

```
use armpl_library

subroutine clasyf(UPLO, N, NB, KB, A, LDA, IPIV, W, LDW, INFO)
```

C specification:

```
#include "armpl.h"

void clasyf_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nb,
             armpl_int_t *kb, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *ipiv,
             armpl_singlecomplex_t *w, const armpl_int_t *ldw,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NB Input parameter.

NB is INTEGER

The maximum number of columns of the matrix A that should be factored. NB should be at least 2 to allow for 2-by-2 pivot blocks.

KB Output parameter.

KB is INTEGER

The number of columns of A that were actually factored. KB is either NB-1 or NB, or N if N <= NB.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, A contains details of the partial factorization.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D.

If UPLO = 'U': Only the last KB elements of IPIV are set.

If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block.

If UPLO = 'L': Only the first KB elements of IPIV are set.

If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

W Output parameter.

W is COMPLEX

W is an array, dimension (LDW, NB) .

LDW Input parameter.

LDW is INTEGER

The leading dimension of the array W. LDW >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit > 0: if INFO = k, D(k,k) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular.

Related Information

For this routine in other precisions, please see *dlasyf*, *slasyf* and *zlasyf*.

4.17.140 clasyf_aa

DLATRF_AA factorizes a panel of a complex symmetric matrix A using the Aasen's algorithm. The panel consists of a set of NB rows of A when UPLO is U, or a set of NB columns when UPLO is L.

In order to factorize the panel, the Aasen's algorithm requires the last row, or column, of the previous panel. The first row, or column, of A is set to be the first row, or column, of an identity matrix, which is used to factorize the first panel.

The resulting J-th row of U, or J-th column of L, is stored in the (J-1)-th row, or column, of A (without the unit diagonals), while the diagonal and subdiagonal of A are overwritten by those of T.

Syntax

Fortran specification:

```
use armpl_library

subroutine clasyf_aa(UPLO, J1, M, NB, A, LDA, IPIV, H, LDH, WORK)
```

C specification:

```
#include "armpl.h"

void clasyf_aa_(const char *uplo, const armpl_int_t *j1, const armpl_int_t *m,
                const armpl_int_t *nb, armpl_singlecomplex_t *a,
                const armpl_int_t *lda, armpl_int_t *ipiv,
                armpl_singlecomplex_t *h, const armpl_int_t *ldh,
                armpl_singlecomplex_t *work, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

J1 Input parameter.

J1 is INTEGER

The location of the first row, or column, of the panel within the submatrix of A, passed to this routine, e.g., when called by CSYTRF_AA, for the first panel, J1 is 1, while for the remaining panels, J1 is 2.

M Input parameter.

M is INTEGER

The dimension of the submatrix. M >= 0.

NB Input parameter.

NB is INTEGER

The dimension of the panel to be facotorized.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, M) for. the first panel, while dimension (LDA,M+1) for the remaining panels.

On entry, A contains the last row, or column, of the previous panel, and the trailing submatrix of A to be factorized, except for the first panel, only the panel is passed.

On exit, the leading panel is factorized.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

IPIV Output parameter.

IPIV is INTEGER array, dimension (M)

Details of the row and column interchanges, the row and column k were interchanged with the row and column IPIV(k).

H Input and output parameter.

H is REAL workspace, dimension (LDH, NB).

LDH Input parameter.

LDH is INTEGER

The leading dimension of the workspace H. LDH >= max(1, M).

WORK Output parameter.

WORK is REAL workspace, dimension (M).

Related Information

For this routine in other precisions, please see *dlasyf_aa*, *slasyf_aa* and *zlasyf_aa*.

4.17.141 clasyf_rook

CLASYF_ROOK computes a partial factorization of a complex symmetric matrix A using the bounded Bunch-Kaufman ("rook") diagonal pivoting method. The partial factorization has the form:

A = (I U12) (A11 0) (I 0) if UPLO = 'U', or:

```
( 0  U22 ) (  0   D  ) ( U12**T U22**T )
```

A = (L11 0) (D 0) ($L11^T$ $L21^T$) if UPLO = 'L'

```
( L21  I ) (  0  A22 ) (  0       I    )
```

where the order of D is at most NB. The actual order is returned in the argument KB, and is either NB or NB-1, or N if N <= NB.

CLASYF_ROOK is an auxiliary routine called by CSYTRF_ROOK. It uses blocked code (calling Level 3 BLAS) to update the submatrix A11 (if UPLO = 'U') or A22 (if UPLO = 'L').

Syntax

Fortran specification:

```
use armpl_library

subroutine clasyf_rook(UPLO, N, NB, KB, A, LDA, IPIV, W, LDW, INFO)
```

C specification:

```
#include "armpl.h"

void clasyf_rook_(const char *uplo, const armpl_int_t *n,
                  const armpl_int_t *nb, armpl_int_t *kb,
                  armpl_singlecomplex_t *a, const armpl_int_t *lda,
                  armpl_int_t *ipiv, armpl_singlecomplex_t *w,
                  const armpl_int_t *ldw, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NB Input parameter.

NB is INTEGER

The maximum number of columns of the matrix A that should be factored. NB should be at least 2 to allow for 2-by-2 pivot blocks.

KB Output parameter.

KB is INTEGER

The number of columns of A that were actually factored. KB is either NB-1 or NB, or N if N <= NB.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, A contains details of the partial factorization.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D.

If UPLO = 'U': Only the last KB elements of IPIV are set.

If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k-1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k-1 and -IPIV(k-1) were inerchaged, D(k-1:k,k-1:k) is a 2-by-2 diagonal block.

If UPLO = 'L': Only the first KB elements of IPIV are set.

If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k+1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k+1 and -IPIV(k+1) were inerchaged, D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

W Output parameter.

W is COMPLEX

W is an array, dimension (LDW, NB) .

LDW Input parameter.

LDW is INTEGER

The leading dimension of the array W. LDW >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit > 0: if INFO = k, D(k,k) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular.

Related Information

For this routine in other precisions, please see *dlasyf_rook*, *slasyf_rook* and *zlasyf_rook*.

4.17.142 clatbs

clatbs solves one of the triangular systems

```
A * x = s*b,  A**T * x = s*b,  or  A**H * x = s*b,
```

with scaling to prevent overflow, where A is an upper or lower triangular band matrix. Here A^T denotes the transpose of A, x and b are n-element vectors, and s is a scaling factor, usually less than or equal to 1, chosen so that the components of x will be less than the overflow threshold. If the unscaled problem will not cause overflow, the Level 2 BLAS routine CTBSV is called. If the matrix A is singular (A(j,j) = 0 for some j), then s is set to 0 and a non-trivial solution to A*x = 0 is returned.

Syntax

Fortran specification:

```
use armpl_library

subroutine clatbs(UPLO, TRANS, DIAG, NORMIN, N, KD, AB, LDAB, X, SCALE, CNORM,
                  INFO)
```

C specification:

```
#include "armpl.h"

void clatbs_(const char *uplo, const char *trans, const char *diag,
             const char *normin, const armpl_int_t *n, const armpl_int_t *kd,
             const armpl_singlecomplex_t *ab, const armpl_int_t *ldab,
             armpl_singlecomplex_t *x, float *scale, float *cnorm,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the matrix A is upper or lower triangular. = 'U': Upper triangular = 'L': Lower triangular

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the operation applied to A. = 'N': Solve A * x = s*b (No transpose) = 'T': Solve A^T * x = s*b (Transpose) = 'C': Solve A^H * x = s*b (Conjugate transpose)

DIAG Input parameter.

DIAG is CHARACTER*1

Specifies whether or not the matrix A is unit triangular. = 'N': Non-unit triangular = 'U': Unit triangular

NORMIN Input parameter.

NORMIN is CHARACTER*1

Specifies whether CNORM has been set or not. = 'Y': CNORM contains the column norms on entry = 'N': CNORM is not set on entry. On exit, the norms will be computed and stored in CNORM.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of subdiagonals or superdiagonals in the triangular matrix A. KD >= 0.

AB Input parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). The upper or lower triangular band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

X Input and output parameter.

X is COMPLEX

X is an array, dimension (N). On entry, the right hand side b of the triangular system. On exit, X is overwritten by the solution vector x.

SCALE Output parameter.

SCALE is REAL

The scaling factor s for the triangular system A * x = s*b, A^T * x = s*b, or A^H * x = s*b. If SCALE = 0, the matrix A is singular or badly scaled, and the vector x is an exact or approximate solution to A*x = 0.

CNORM Input and output parameter.

CNORM is REAL

CNORM is an array, dimension (N). If NORMIN = 'Y', CNORM is an input argument and CNORM(j) contains the norm of the off-diagonal part of the j-th column of A. If TRANS = 'N', CNORM(j) must be greater than or equal to the infinity-norm, and if TRANS = 'T' or 'C', CNORM(j) must be greater than or equal to the 1-norm.

If NORMIN = 'N', CNORM is an output argument and CNORM(j) returns the 1-norm of the offdiagonal part of the j-th column of A.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dlatbs*, *slatbs* and *zlatbs*.

4.17.143 clatdf

clatdf computes the contribution to the reciprocal Dif-estimate by solving for x in Z * x = b, where b is chosen such that the norm of x is as large as possible. It is assumed that LU decomposition of Z has been computed by CGETC2. On entry RHS = f holds the contribution from earlier solved sub-systems, and on return RHS = x.

The factorization of Z returned by CGETC2 has the form Z = P * L * U * Q, where P and Q are permutation matrices. L is lower triangular with unit diagonal elements and U is upper triangular.

Syntax

Fortran specification:

```
use armpl_library

subroutine clatdf(IJOB, N, Z, LDZ, RHS, RDSUM, RDSCAL, IPIV, JPIV)
```

C specification:

```
#include "armpl.h"

void clatdf_(const armpl_int_t *ijob, const armpl_int_t *n,
             const armpl_singlecomplex_t *z, const armpl_int_t *ldz,
             armpl_singlecomplex_t *rhs, float *rdsum, float *rdscal,
             const armpl_int_t *ipiv, const armpl_int_t *jpiv);
```

Parameters

IJOB Input parameter.

IJOB is INTEGER

IJOB = 2: First compute an approximative null-vector e of Z using CGECON, e is normalized and solve for Zx = +-e - f with the sign giving the greater value of 2-norm(x). About 5 times as expensive as Default. IJOB .ne. 2: Local look ahead strategy where all entries of the r.h.s. b is chosen as either +1 or -1. Default.

N Input parameter.

N is INTEGER

The number of columns of the matrix Z.

Z Input parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, N). On entry, the LU part of the factorization of the n-by-n matrix Z computed by CGETC2: Z = P * L * U * Q

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDA >= max(1, N).

RHS Input and output parameter.

RHS is COMPLEX

RHS is an array, dimension (N).. On entry, RHS contains contributions from other subsystems. On exit, RHS contains the solution of the subsystem with entries according to the value of IJOB (see above).

RDSUM Input and output parameter.

RDSUM is REAL

On entry, the sum of squares of computed contributions to the Dif-estimate under computation by CTGSYL, where the scaling factor RDSCAL (see below) has been factored out. On exit, the corresponding sum of squares updated with the contributions from the current sub-system. If TRANS = 'T' RDSUM is not touched. NOTE: RDSUM only makes sense when CTGSY2 is called by CTGSYL.

RDSCAL Input and output parameter.

RDSCAL is REAL

On entry, scaling factor used to prevent overflow in RDSUM. On exit, RDSCAL is updated w.r.t. the current contributions in RDSUM. If TRANS = 'T', RDSCAL is not touched. NOTE: RDSCAL only makes sense when CTGSY2 is called by CTGSYL.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N).

The pivot indices; for 1 <= i <= N, row i of the matrix has been interchanged with row IPIV(i).

JPIV Input parameter.

JPIV is INTEGER array, dimension (N).

The pivot indices; for 1 <= j <= N, column j of the matrix has been interchanged with column JPIV(j).

Related Information

For this routine in other precisions, please see *dlatdf*, *slatdf* and *zlatdf*.

4.17.144 clatps

clatps solves one of the triangular systems

```
A * x = s*b,  A**T * x = s*b,  or  A**H * x = s*b,
```

with scaling to prevent overflow, where A is an upper or lower triangular matrix stored in packed form. Here A^T denotes the transpose of A, A^H denotes the conjugate transpose of A, x and b are n-element vectors, and s is a scaling factor, usually less than or equal to 1, chosen so that the components of x will be less than the overflow threshold. If the unscaled problem will not cause overflow, the Level 2 BLAS routine CTPSV is called. If the matrix A is singular (A(j,j) = 0 for some j), then s is set to 0 and a non-trivial solution to A*x = 0 is returned.

Syntax

Fortran specification:

```
use armpl_library

subroutine clatps(UPLO, TRANS, DIAG, NORMIN, N, AP, X, SCALE, CNORM, INFO)
```

C specification:

```
#include "armpl.h"

void clatps_(const char *uplo, const char *trans, const char *diag,
             const char *normin, const armpl_int_t *n,
             const armpl_singlecomplex_t *ap, armpl_singlecomplex_t *x,
             float *scale, float *cnorm, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the matrix A is upper or lower triangular. = 'U': Upper triangular = 'L': Lower triangular

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the operation applied to A. = 'N': Solve A * x = s*b (No transpose) = 'T': Solve A^T * x = s*b (Transpose) = 'C': Solve A^H * x = s*b (Conjugate transpose)

DIAG Input parameter.

DIAG is CHARACTER*1

Specifies whether or not the matrix A is unit triangular. = 'N': Non-unit triangular = 'U': Unit triangular

NORMIN Input parameter.

NORMIN is CHARACTER*1

Specifies whether CNORM has been set or not. = 'Y': CNORM contains the column norms on entry = 'N': CNORM is not set on entry. On exit, the norms will be computed and stored in CNORM.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2). The upper or lower triangular matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

X Input and output parameter.

X is COMPLEX

X is an array, dimension (N). On entry, the right hand side b of the triangular system. On exit, X is overwritten by the solution vector x.

SCALE Output parameter.

SCALE is REAL

The scaling factor s for the triangular system A * x = s*b, A^T * x = s*b, or A^H * x = s*b. If SCALE = 0, the matrix A is singular or badly scaled, and the vector x is an exact or approximate solution to A*x = 0.

CNORM Input and output parameter.

CNORM is REAL

CNORM is an array, dimension (N). If NORMIN = 'Y', CNORM is an input argument and CNORM(j) contains the norm of the off-diagonal part of the j-th column of A. If TRANS = 'N', CNORM(j) must be greater than or equal to the infinity-norm, and if TRANS = 'T' or 'C', CNORM(j) must be greater than or equal to the 1-norm.

If NORMIN = 'N', CNORM is an output argument and CNORM(j) returns the 1-norm of the offdiagonal part of the j-th column of A.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dlatps*, *slatps* and *zlatps*.

4.17.145 clatrd

clatrd reduces NB rows and columns of a complex Hermitian matrix A to Hermitian tridiagonal form by a unitary similarity transformation Q^H * A * Q, and returns the matrices V and W which are needed to apply the transformation to the unreduced part of A.

If UPLO = 'U', clatrd reduces the last NB rows and columns of a matrix, of which the upper triangle is supplied; if UPLO = 'L', clatrd reduces the first NB rows and columns of a matrix, of which the lower triangle is supplied.

This is an auxiliary routine called by CHETRD.

Syntax

Fortran specification:

```
use armpl_library

subroutine clatrd(UPLO, N, NB, A, LDA, E, TAU, W, LDW)
```

C specification:

```
#include "armpl.h"

void clatrd_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nb,
             armpl_singlecomplex_t *a, const armpl_int_t *lda, float *e,
             armpl_singlecomplex_t *tau, armpl_singlecomplex_t *w,
             const armpl_int_t *ldw, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the Hermitian matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A.

NB Input parameter.

NB is INTEGER

The number of rows and columns to be reduced.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit: if UPLO = 'U', the last NB columns have been reduced to tridiagonal form, with the diagonal elements overwriting the diagonal elements of A; the elements above the diagonal with the array TAU, represent the unitary matrix Q as a product of elementary reflectors; if UPLO = 'L', the first NB columns have been reduced to tridiagonal form, with the diagonal elements overwriting the diagonal elements of A; the elements below the diagonal with the array TAU, represent the unitary matrix Q as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Output parameter.

E is REAL

E is an array, dimension (N-1). If UPLO = 'U', E(n-nb:n-1) contains the superdiagonal elements of the last NB columns of the reduced matrix; if UPLO = 'L', E(1:nb) contains the subdiagonal elements of the first NB columns of the reduced matrix.

TAU Output parameter.

TAU is COMPLEX

TAU is an array, dimension (N-1). The scalar factors of the elementary reflectors, stored in TAU(n-nb:n-1) if UPLO = 'U', and in TAU(1:nb) if UPLO = 'L'. See Further Details.

W Output parameter.

W is COMPLEX

W is an array, dimension (LDW, NB). The n-by-nb matrix W required to update the unreduced part of A.

LDW Input parameter.

LDW is INTEGER

The leading dimension of the array W. LDW >= max(1, N).

Related Information

For this routine in other precisions, please see *dlatrd*, *slatrd* and *zlatrd*.

4.17.146 clatrs

clatrs solves one of the triangular systems

```
A * x = s*b,  A**T * x = s*b,  or  A**H * x = s*b,
```

with scaling to prevent overflow. Here A is an upper or lower triangular matrix, A^T denotes the transpose of A, A^H denotes the conjugate transpose of A, x and b are n-element vectors, and s is a scaling factor, usually less than or equal to 1, chosen so that the components of x will be less than the overflow threshold. If the unscaled problem will not cause overflow, the Level 2 BLAS routine CTRSV is called. If the matrix A is singular (A(j,j) = 0 for some j), then s is set to 0 and a non-trivial solution to A*x = 0 is returned.

Syntax

Fortran specification:

```
use armpl_library

subroutine clatrs(UPLO, TRANS, DIAG, NORMIN, N, A, LDA, X, SCALE, CNORM,
                  INFO)
```

C specification:

```
#include "armpl.h"

void clatrs_(const char *uplo, const char *trans, const char *diag,
             const char *normin, const armpl_int_t *n,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda,
             armpl_singlecomplex_t *x, float *scale, float *cnorm,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the matrix A is upper or lower triangular. = 'U': Upper triangular = 'L': Lower triangular

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the operation applied to A. = 'N': Solve A * x = s*b (No transpose) = 'T': Solve A^T * x = s*b (Transpose) = 'C': Solve A^H * x = s*b (Conjugate transpose)

DIAG Input parameter.

DIAG is CHARACTER*1

Specifies whether or not the matrix A is unit triangular. = 'N': Non-unit triangular = 'U': Unit triangular

NORMIN Input parameter.

NORMIN is CHARACTER*1

Specifies whether CNORM has been set or not. = 'Y': CNORM contains the column norms on entry = 'N': CNORM is not set on entry. On exit, the norms will be computed and stored in CNORM.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The triangular matrix A. If UPLO = 'U', the leading n by n upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n by n lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max (1, N).

X Input and output parameter.

X is COMPLEX

X is an array, dimension (N). On entry, the right hand side b of the triangular system. On exit, X is overwritten by the solution vector x.

SCALE Output parameter.

SCALE is REAL

The scaling factor s for the triangular system A * x = s*b, A^T * x = s*b, or A^H * x = s*b. If SCALE = 0, the matrix A is singular or badly scaled, and the vector x is an exact or approximate solution to A*x = 0.

CNORM Input and output parameter.

CNORM is REAL

CNORM is an array, dimension (N). If NORMIN = 'Y', CNORM is an input argument and CNORM(j) contains the norm of the off-diagonal part of the j-th column of A. If TRANS = 'N', CNORM(j) must be greater than or equal to the infinity-norm, and if TRANS = 'T' or 'C', CNORM(j) must be greater than or equal to the 1-norm.

If NORMIN = 'N', CNORM is an output argument and CNORM(j) returns the 1-norm of the offdiagonal part of the j-th column of A.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dlatrs*, *slatrs* and *zlatrs*.

4.17.147 clatrz

clatrz factors the M-by-(M+L) complex upper trapezoidal matrix [A1 A2] = [A(1:M,1:M) A(1:M,N-L+1:N)] as (R 0) * Z by means of unitary transformations, where Z is an (M+L)-by-(M+L) unitary matrix and, R and A1 are M-by-M upper triangular matrices.

Syntax

Fortran specification:

```
use armpl_library

subroutine clatrz(M, N, L, A, LDA, TAU, WORK)
```

C specification:

```
#include "armpl.h"

void clatrz_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *l,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             armpl_singlecomplex_t *tau, armpl_singlecomplex_t *work);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

L Input parameter.

L is INTEGER

The number of columns of the matrix A containing the meaningful part of the Householder vectors. N-M >= L >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the leading M-by-N upper trapezoidal part of the array A must contain the matrix to be factorized. On exit, the leading M-by-M upper triangular part of A contains the upper triangular matrix R, and elements N-L+1 to N of the first M rows of A, with the array TAU, represent the unitary matrix Z as a product of M elementary reflectors.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is COMPLEX

TAU is an array, dimension (M). The scalar factors of the elementary reflectors.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (M) .

Related Information

For this routine in other precisions, please see *dlatrz*, *slatrz* and *zlatrz*.

4.17.148 clatsqr

SLATSQR computes a blocked Tall-Skinny QR factorization of an M-by-N matrix A, where M >= N: A = Q * R .

Syntax

Fortran specification:

```
use armpl_library

subroutine clatsqr(M, N, MB, NB, A, LDA, T, LDT, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void clatsqr_(const armpl_int_t *m, const armpl_int_t *n,
              const armpl_int_t *mb, const armpl_int_t *nb,
              armpl_singlecomplex_t *a, const armpl_int_t *lda,
              armpl_singlecomplex_t *t, const armpl_int_t *ldt,
              armpl_singlecomplex_t *work, const armpl_int_t *lwork,
              armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. M >= N >= 0.

MB Input parameter.

MB is INTEGER

The row block size to be used in the blocked QR. MB > N.

NB Input parameter.

NB is INTEGER

The column block size to be used in the blocked QR. N >= NB >= 1.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the N-by-N upper triangular matrix R; the elements below the diagonal represent Q by the columns of blocked V (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

T Output parameter.

T is COMPLEX

T is an array,. dimension (LDT, N * Number_of_row_blocks) where Number_of_row_blocks = CEIL((M-N)/(MB-N)) The blocked upper triangular block reflectors stored in compact form as a sequence of upper triangular blocks. See Further Details below.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= NB.

WORK Output parameter.

(workspace) COMPLEX

(workspace) is an array, dimension (MAX(1, LWORK)) .

LWORK Input parameter.

The dimension of the array WORK. LWORK >= NB*N.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dlatsqr*, *slatsqr* and *zlatsqr*.

4.17.149 clauu2

clauu2 computes the product U * U^H or L^H * L, where the triangular factor U or L is stored in the upper or lower triangular part of the array A.

If UPLO = 'U' or 'u' then the upper triangle of the result is stored, overwriting the factor U in A. If UPLO = 'L' or 'l' then the lower triangle of the result is stored, overwriting the factor L in A.

This is the unblocked form of the algorithm, calling Level 2 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine clauu2(UPLO, N, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void clauu2_(const char *uplo, const armpl_int_t *n, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the triangular factor stored in the array A is upper or lower triangular: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the triangular factor U or L. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product U * U^H ; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product L^H * L.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dlauu2*, *slauu2* and *zlauu2*.

4.17.150 clauum

clauum computes the product U * U^H or L^H * L, where the triangular factor U or L is stored in the upper or lower triangular part of the array A.

If UPLO = 'U' or 'u' then the upper triangle of the result is stored, overwriting the factor U in A. If UPLO = 'L' or 'l' then the lower triangle of the result is stored, overwriting the factor L in A.

This is the blocked form of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine clauum(UPLO, N, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void clauum_(const char *uplo, const armpl_int_t *n, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the triangular factor stored in the array A is upper or lower triangular: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the triangular factor U or L. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product U * U^H ; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product L^H * L.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dlauum*, *slauum* and *zlauum*. It also exists with a native C interface as *LAPACKE_clauum*.

4.17.151 cpbtf2

cpbtf2 computes the Cholesky factorization of a complex Hermitian positive definite band matrix A.

The factorization has the form

```
 A = U**H * U ,  if UPLO = 'U', or
A = L  * L**H,  if UPLO = 'L',
```

where U is an upper triangular matrix, U^H is the conjugate transpose of U, and L is lower triangular.

This is the unblocked version of the algorithm, calling Level 2 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine cpbtf2(UPLO, N, KD, AB, LDAB, INFO)
```

C specification:

```
#include "armpl.h"

void cpbtf2_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
             armpl_singlecomplex_t *ab, const armpl_int_t *ldab,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the Hermitian matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of super-diagonals of the matrix A if UPLO = 'U', or the number of sub-diagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is COMPLEX

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the Hermitian band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, if INFO = 0, the triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H of the band matrix A, in the same storage format as A.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value > 0: if INFO = k, the leading minor of order k is not positive definite, and the factorization could not be completed.

Related Information

For this routine in other precisions, please see *dpbtf2*, *spbtf2* and *zpbtf2*.

4.17.152 cpotf2

cpotf2 computes the Cholesky factorization of a complex Hermitian positive definite matrix A.

The factorization has the form

```
 A = U**H * U ,  if UPLO = 'U', or
A = L  * L**H,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is lower triangular.

This is the unblocked version of the algorithm, calling Level 2 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine cpotf2(UPLO, N, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void cpotf2_(const char *uplo, const armpl_int_t *n, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the Hermitian matrix A is stored. = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading n by n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n by n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H .

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value > 0: if INFO = k, the leading minor of order k is not positive definite, and the factorization could not be completed.

Related Information

For this routine in other precisions, please see *dpotf2*, *spotf2* and *zpotf2*.

4.17.153 cpstf2

cpstf2 computes the Cholesky factorization with complete pivoting of a complex Hermitian positive semidefinite matrix A.

The factorization has the form

```
 P**T * A * P = U**H * U ,  if UPLO = 'U',
P**T * A * P = L  * L**H,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is lower triangular, and P is stored as vector PIV.

This algorithm does not attempt to check that A is positive semidefinite. This version of the algorithm calls level 2 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine cpstf2(UPLO, N, A, LDA, PIV, RANK, TOL, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void cpstf2_(const char *uplo, const armpl_int_t *n, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *piv, armpl_int_t *rank,
             const float *tol, float *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored. = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading n by n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n by n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization as above.

PIV Output parameter.

PIV is INTEGER array, dimension (N)

PIV is such that the nonzero entries are P(PIV(K), K) = 1.

RANK Output parameter.

RANK is INTEGER

The rank of A given by the number of steps the algorithm completed.

TOL Input parameter.

TOL is REAL

User defined tolerance. If TOL < 0, then N*U*MAX(A(K,K)) will be used. The algorithm terminates at the (K-1)st step if the pivot <= TOL.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (2*N). Work space.

INFO Output parameter.

INFO is INTEGER

< 0: If INFO = -K, the K-th argument had an illegal value, = 0: algorithm completed successfully, and > 0: the matrix A is either rank deficient with computed rank as returned in RANK, or is not positive semidefinite. See Section 7 of LAPACK Working Note #161 for further information.

Related Information

For this routine in other precisions, please see *dpstf2*, *spstf2* and *zpstf2*.

4.17.154 cptts2

cptts2 solves a tridiagonal system of the form

```
A * X = B
```

using the factorization A = U^H *D*U or A = L*D*L^H computed by CPTTRF. D is a diagonal matrix specified in the vector D, U (or L) is a unit bidiagonal matrix whose superdiagonal (subdiagonal) is specified in the vector E, and X and B are N by NRHS matrices.

Syntax

Fortran specification:

```
use armpl_library

subroutine cptts2(IUPLO, N, NRHS, D, E, B, LDB)
```

C specification:

```
#include "armpl.h"

void cptts2_(const armpl_int_t *iuplo, const armpl_int_t *n,
             const armpl_int_t *nrhs, const float *d,
             const armpl_singlecomplex_t *e, armpl_singlecomplex_t *b,
             const armpl_int_t *ldb);
```

Parameters

IUPLO Input parameter.

IUPLO is INTEGER

Specifies the form of the factorization and whether the vector E is the superdiagonal of the upper bidiagonal factor U or the subdiagonal of the lower bidiagonal factor L. = 1: A = U^H *D*U, E is the superdiagonal of U = 0: A = L*D*L^H , E is the subdiagonal of L

N Input parameter.

N is INTEGER

The order of the tridiagonal matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

D Input parameter.

D is REAL

D is an array, dimension (N). The n diagonal elements of the diagonal matrix D from the factorization A = U^H *D*U or A = L*D*L^H .

E Input parameter.

E is COMPLEX

E is an array, dimension (N-1). If IUPLO = 1, the (n-1) superdiagonal elements of the unit bidiagonal factor U from the factorization A = U^H *D*U. If IUPLO = 0, the (n-1) subdiagonal elements of the unit bidiagonal factor L from the factorization A = L*D*L^H .

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, NRHS). On entry, the right hand side vectors B for the system of linear equations. On exit, the solution vectors, X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Related Information

For this routine in other precisions, please see *dptts2*, *sptts2* and *zptts2*.

4.17.155 csrscl

csrscl multiplies an n-element complex vector x by the real scalar 1/a. This is done without overflow or underflow as long as the final result x/a does not overflow or underflow.

Syntax

Fortran specification:

```
use armpl_library

subroutine csrscl(N, SA, SX, INCX)
```

C specification:

```
#include "armpl.h"

void csrscl_(const armpl_int_t *n, const float *sa, armpl_singlecomplex_t *sx,
             const armpl_int_t *incx);
```

Parameters

N Input parameter.

N is INTEGER

The number of components of the vector x.

SA Input parameter.

SA is REAL

The scalar a which is used to divide each component of x. SA must be >= 0, or the subroutine will divide by zero.

SX Input and output parameter.

SX is COMPLEX

SX is an array, dimension. (1+(N-1)*abs(INCX)) The n-element vector x.

INCX Input parameter.

INCX is INTEGER

The increment between successive values of the vector SX. > 0: SX(1) = X(1) and SX(1+(i-1)*INCX) = x(i), 1< i<= n

Related Information

4.17.156 csyconv

csyconv convert A given by TRF into L and D and vice-versa. Get Non-diag elements of D (returned in workspace) and apply or reverse permutation done in TRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine csyconv(UPLO, WAY, N, A, LDA, IPIV, E, INFO)
```

C specification:

```
#include "armpl.h"

void csyconv_(const char *uplo, const char *way, const armpl_int_t *n,
              armpl_singlecomplex_t *a, const armpl_int_t *lda,
              const armpl_int_t *ipiv, armpl_singlecomplex_t *e,
              armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

WAY Input parameter.

WAY is CHARACTER*1

= 'C': Convert = 'R': Revert

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CSYTRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CSYTRF.

E Output parameter.

E is COMPLEX

E is an array, dimension (N). E stores the supdiagonal/subdiagonal of the symmetric 1-by-1 or 2-by-2 block diagonal matrix D in LDLT.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dsyconv*, *ssyconv* and *zsyconv*. It also exists with a native C interface as *LAPACKE_csyconv*.

4.17.157 csyswapr

CSYSWAPR applies an elementary permutation on the rows and the columns of a symmetric matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine csyswapr(UPLO, N, A, LDA, I1, I2)
```

C specification:

```
#include "armpl.h"

void csyswapr_(const char *uplo, const armpl_int_t *n,
               armpl_singlecomplex_t *a, const armpl_int_t *lda,
               const armpl_int_t *i1, const armpl_int_t *i2, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the NB diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CSYTRF.

On exit, if INFO = 0, the (symmetric) inverse of the original matrix. If UPLO = 'U', the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; if UPLO = 'L' the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

I1 Input parameter.

I1 is INTEGER

Index of the first row to swap

I2 Input parameter.

I2 is INTEGER

Index of the second row to swap

Related Information

For this routine in other precisions, please see *dsyswapr*, *ssyswapr* and *zsyswapr*. It also exists with a native C interface as *LAPACKE_csyswapr*.

4.17.158 csytf2

csytf2 computes the factorization of a complex symmetric matrix A using the Bunch-Kaufman diagonal pivoting method:

```
A = U*D*U**T  or  A = L*D*L**T
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, U^T is the transpose of U, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the unblocked version of the algorithm, calling Level 2 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine csytf2(UPLO, N, A, LDA, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void csytf2_(const char *uplo, const armpl_int_t *n, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *ipiv, armpl_int_t *info,
             ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L (see below for further details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D.

If UPLO = 'U': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block.

If UPLO = 'L': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value > 0: if INFO = k, D(k,k) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *dsytf2*, *ssytf2* and *zsytf2*.

4.17.159 csytf2_rook

CSYTF2_ROOK computes the factorization of a complex symmetric matrix A using the bounded Bunch-Kaufman ("rook") diagonal pivoting method:

```
A = U*D*U**T  or  A = L*D*L**T
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, U^T is the transpose of U, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the unblocked version of the algorithm, calling Level 2 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine csytf2_rook(UPLO, N, A, LDA, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void csytf2_rook_(const char *uplo, const armpl_int_t *n,
                  armpl_singlecomplex_t *a, const armpl_int_t *lda,
                  armpl_int_t *ipiv, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L (see below for further details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D.

If UPLO = 'U': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k-1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k-1 and -IPIV(k-1) were inerchaged, D(k-1:k,k-1:k) is a 2-by-2 diagonal block.

If UPLO = 'L': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k+1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k+1 and -IPIV(k+1) were inerchaged, D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value > 0: if INFO = k, D(k,k) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *dsytf2_rook*, *ssytf2_rook* and *zsytf2_rook*.

4.17.160 ctfsm

Level 3 BLAS like routine for A in RFP Format.

ctfsm solves the matrix equation

```
op( A )*X = alpha*B  or  X*op( A ) = alpha*B
```

where alpha is a scalar, X and B are m by n matrices, A is a unit, or non-unit, upper or lower triangular matrix and op(A) is one of

```
op( A ) = A   or   op( A ) = A**H.
```

A is in Rectangular Full Packed (RFP) Format.

The matrix X is overwritten on B.

Syntax

Fortran specification:

```
use armpl_library

subroutine ctfsm(TRANSR, SIDE, UPLO, TRANS, DIAG, M, N, ALPHA, A, B, LDB)
```

C specification:

```
#include "armpl.h"

void ctfsm_(const char *transr, const char *side, const char *uplo,
            const char *trans, const char *diag, const armpl_int_t *m,
            const armpl_int_t *n, const armpl_singlecomplex_t *alpha,
            const armpl_singlecomplex_t *a, armpl_singlecomplex_t *b,
            const armpl_int_t *ldb, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': The Normal Form of RFP A is stored; = 'C': The Conjugate-transpose Form of RFP A is stored.

SIDE Input parameter.

SIDE is CHARACTER*1

On entry, SIDE specifies whether op(A) appears on the left or right of X as follows:

SIDE = 'L' or 'l' op(A)*X = alpha*B.

SIDE = 'R' or 'r' X*op(A) = alpha*B.

Unchanged on exit.

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the RFP matrix A came from an upper or lower triangular matrix as follows: UPLO = 'U' or 'u' RFP A came from an upper triangular matrix UPLO = 'L' or 'l' RFP A came from a lower triangular matrix

Unchanged on exit.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the form of op(A) to be used in the matrix multiplication as follows:

TRANS = 'N' or 'n' op(A) = A.

TRANS = 'C' or 'c' op(A) = conjg(A').

Unchanged on exit.

DIAG Input parameter.

DIAG is CHARACTER*1

On entry, DIAG specifies whether or not RFP A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

Unchanged on exit.

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of B. M must be at least zero. Unchanged on exit.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of B. N must be at least zero. Unchanged on exit.

ALPHA Input parameter.

ALPHA is COMPLEX

On entry, ALPHA specifies the scalar alpha. When alpha is zero then A is not referenced and B need not be set before entry. Unchanged on exit.

A Input parameter.

A is COMPLEX

A is an array, dimension (N*(N+1)/2). NT = N*(N+1)/2. On entry, the matrix A in RFP Format. RFP Format is described by TRANSR, UPLO and N as follows: If TRANSR='N' then RFP A is (0:N,0:K-1) when N is even; K=N/2. RFP A is (0:N-1,0:K) when N is odd; K=N/2. If TRANSR = 'C' then RFP is the Conjugate-transpose of RFP A as defined when TRANSR = 'N'. The contents of RFP A are defined by UPLO as follows: If UPLO = 'U' the RFP A contains the NT elements of upper packed A either in normal or conjugate-transpose Format. If UPLO = 'L' the RFP A contains the NT elements of lower packed A either in normal or conjugate-transpose Format. The LDA of RFP A is (N+1)/2 when TRANSR = 'C'. When TRANSR is 'N' the LDA is N+1 when N is even and is N when is odd. See the Note below for more details. Unchanged on exit.

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, N). Before entry, the leading m by n part of the array B must contain the right-hand side matrix B, and on exit is overwritten by the solution matrix X.

LDB Input parameter.

LDB is INTEGER

On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. LDB must be at least max(1, m). Unchanged on exit.

Related Information

For this routine in other precisions, please see *dtfsm*, *stfsm* and *ztfsm*. It also exists with a native C interface as *LAPACKE_ctfsm*.

4.17.161 ctfttp

ctfttp copies a triangular matrix A from rectangular full packed format (TF) to standard packed format (TP).

Syntax

Fortran specification:

```
use armpl_library

subroutine ctfttp(TRANSR, UPLO, N, ARF, AP, INFO)
```

C specification:

```
#include "armpl.h"

void ctfttp_(const char *transr, const char *uplo, const armpl_int_t *n,
             const armpl_singlecomplex_t *arf, armpl_singlecomplex_t *ap,
             armpl_int_t *info, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': ARF is in Normal format; = 'C': ARF is in Conjugate-transpose format;

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

ARF Input parameter.

ARF is COMPLEX

ARF is an array, dimension (N*(N+1)/2),. On entry, the upper or lower triangular matrix A stored in RFP format. For a further discussion see Notes below.

AP Output parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2),. On exit, the upper or lower triangular matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dtfttp*, *stfttp* and *ztfttp*. It also exists with a native C interface as *LAPACKE_ctfttp*.

4.17.162 ctfttr

ctfttr copies a triangular matrix A from rectangular full packed format (TF) to standard full format (TR).

Syntax

Fortran specification:

```
use armpl_library

subroutine ctfttr(TRANSR, UPLO, N, ARF, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void ctfttr_(const char *transr, const char *uplo, const armpl_int_t *n,
             const armpl_singlecomplex_t *arf, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *info, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': ARF is in Normal format; = 'C': ARF is in Conjugate-transpose format;

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

ARF Input parameter.

ARF is COMPLEX

ARF is an array, dimension (N*(N+1)/2),. On entry, the upper or lower triangular matrix A stored in RFP format. For a further discussion see Notes below.

A Output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On exit, the triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dtfttr*, *stfttr* and *ztfttr*. It also exists with a native C interface as *LAPACKE_ctfttr*.

4.17.163 ctgex2

`ctgex2` swaps adjacent diagonal 1 by 1 blocks (A11,B11) and (A22,B22) in an upper triangular matrix pair (A, B) by an unitary equivalence transformation.

(A, B) must be in generalized Schur canonical form, that is, A and B are both upper triangular.

Optionally, the matrices Q and Z of generalized Schur vectors are updated.

```
Q(in) * A(in) * Z(in)**H = Q(out) * A(out) * Z(out)**H
Q(in) * B(in) * Z(in)**H = Q(out) * B(out) * Z(out)**H
```

Syntax

Fortran specification:

```
use armpl_library

subroutine ctgex2(WANTQ, WANTZ, N, A, LDA, B, LDB, Q, LDQ, Z, LDZ, J1, INFO)
```

C specification:

```
#include "armpl.h"

void ctgex2_(const armpl_int_t *wantq, const armpl_int_t *wantz,
             const armpl_int_t *n, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, armpl_singlecomplex_t *q,
             const armpl_int_t *ldq, armpl_singlecomplex_t *z,
             const armpl_int_t *ldz, const armpl_int_t *j1,
             armpl_int_t *info);
```

Parameters

WANTQ Input parameter.

`WANTQ` is `LOGICAL`

.TRUE. : update the left transformation matrix Q; .FALSE.: do not update `Q`.

WANTZ Input parameter.

`WANTZ` is `LOGICAL`

.TRUE. : update the right transformation matrix Z; .FALSE.: do not update `Z`.

N Input parameter.

`N` is `INTEGER`

The order of the matrices `A` and `B`. `N` >= 0.

A Input and output parameter.

`A` is `COMPLEX`

`A` is an array, dimension (`LDA`, `N`). On entry, the matrix `A` in the pair (`A`, B). On exit, the updated matrix `A`.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, N). On entry, the matrix B in the pair (A, B). On exit, the updated matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Q Input and output parameter.

Q is COMPLEX

Q is an array, dimension (LDQ, N). If WANTQ = .TRUE, on entry, the unitary matrix Q. On exit, the updated matrix Q. Not referenced if WANTQ = .FALSE..

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= 1; If WANTQ = .TRUE., LDQ >= N.

Z Input and output parameter.

Z is COMPLEX

Z is an array, dimension (LDZ, N). If WANTZ = .TRUE, on entry, the unitary matrix Z. On exit, the updated matrix Z. Not referenced if WANTZ = .FALSE..

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1; If WANTZ = .TRUE., LDZ >= N.

J1 Input parameter.

J1 is INTEGER

The index to the first block (A11, B11).

INFO Output parameter.

INFO is INTEGER

=0: Successful exit. =1: The transformed matrix pair (A, B) would be too far from generalized Schur form; the problem is ill- conditioned.

Related Information

For this routine in other precisions, please see *dtgex2*, *stgex2* and *ztgex2*.

4.17.164 ctgsy2

ctgsy2 solves the generalized Sylvester equation

```
A * R - L * B = scale *  C                (1)
D * R - L * E = scale * F
```

using Level 1 and 2 BLAS, where R and L are unknown M-by-N matrices, (A, D), (B, E) and (C, F) are given matrix pairs of size M-by-M, N-by-N and M-by-N, respectively. A, B, D and E are upper triangular (i.e., (A,D) and (B,E) in generalized Schur form).

The solution (R, L) overwrites (C, F). 0 <= SCALE <= 1 is an output scaling factor chosen to avoid overflow.

In matrix notation solving equation (1) corresponds to solve Zx = scale * b, where Z is defined as

```
Z = [ kron(In, A)  -kron(B**H, Im) ]             (2)
    [ kron(In, D)  -kron(E**H, Im) ],
```

Ik is the identity matrix of size k and X^H is the transpose of X. kron(X, Y) is the Kronecker product between the matrices X and Y.

If TRANS = 'C', y in the conjugate transposed system Z^H *y = scale*b is solved for, which is equivalent to solve for R and L in

```
 A**H * R  + D**H * L   = scale * C           (3)
R  * B**H + L  * E**H  = scale * -F
```

This case is used to compute an estimate of Dif[(A, D), (B, E)] = = sigma_min(Z) using reverse communicaton with CLACON.

`ctgsy2` also (IJOB >= 1) contributes to the computation in CTGSYL of an upper bound on the separation between to matrix pairs. Then the input (A, D), (B, E) are sub-pencils of two matrix pairs in CTGSYL.

Syntax

Fortran specification:

```
use armpl_library

subroutine ctgsy2(TRANS, IJOB, M, N, A, LDA, B, LDB, C, LDC, D, LDD, E, LDE,
                  F, LDF, SCALE, RDSUM, RDSCAL, INFO)
```

C specification:

```
#include "armpl.h"

void ctgsy2_(const char *trans, const armpl_int_t *ijob, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_singlecomplex_t *a,
             const armpl_int_t *lda, const armpl_singlecomplex_t *b,
             const armpl_int_t *ldb, armpl_singlecomplex_t *c,
             const armpl_int_t *ldc, const armpl_singlecomplex_t *d,
             const armpl_int_t *ldd, const armpl_singlecomplex_t *e,
             const armpl_int_t *lde, armpl_singlecomplex_t *f,
             const armpl_int_t *ldf, float *scale, float *rdsum,
             float *rdscal, armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N', solve the generalized Sylvester equation (1). = 'T': solve the 'transposed' system (3).

IJOB Input parameter.

IJOB is INTEGER

Specifies what kind of functionality to be performed. =0: solve (1) only. =1: A contribution from this subsystem to a Frobenius norm-based estimate of the separation between two matrix pairs is computed. (look

ahead strategy is used). =2: A contribution from this subsystem to a Frobenius norm-based estimate of the separation between two matrix pairs is computed. (SGECON on sub-systems is used.) Not referenced if TRANS = 'T'.

M Input parameter.

M is INTEGER

On entry, M specifies the order of A and D, and the row dimension of C, F, R and L.

N Input parameter.

N is INTEGER

On entry, N specifies the order of B and E, and the column dimension of C, F, R and L.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, M). On entry, A contains an upper triangular matrix.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the matrix A. LDA >= max(1, M).

B Input parameter.

B is COMPLEX

B is an array, dimension (LDB, N). On entry, B contains an upper triangular matrix.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the matrix B. LDB >= max(1, N).

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). On entry, C contains the right-hand-side of the first matrix equation in (1). On exit, if IJOB = 0, C has been overwritten by the solution R.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the matrix C. LDC >= max(1, M).

D Input parameter.

D is COMPLEX

D is an array, dimension (LDD, M). On entry, D contains an upper triangular matrix.

LDD Input parameter.

LDD is INTEGER

The leading dimension of the matrix D. LDD >= max(1, M).

E Input parameter.

E is COMPLEX

E is an array, dimension (LDE, N). On entry, E contains an upper triangular matrix.

LDE Input parameter.

LDE is INTEGER

The leading dimension of the matrix E. LDE >= max(1, N).

F Input and output parameter.

F is COMPLEX

F is an array, dimension (LDF, N). On entry, F contains the right-hand-side of the second matrix equation in (1). On exit, if IJOB = 0, F has been overwritten by the solution L.

LDF Input parameter.

LDF is INTEGER

The leading dimension of the matrix F. LDF >= max(1, M).

SCALE Output parameter.

SCALE is REAL

On exit, 0 <= SCALE <= 1. If 0 < SCALE < 1, the solutions R and L (C and F on entry) will hold the solutions to a slightly perturbed system but the input matrices A, B, D and E have not been changed. If SCALE = 0, R and L will hold the solutions to the homogeneous system with C = F = 0. Normally, SCALE = 1.

RDSUM Input and output parameter.

RDSUM is REAL

On entry, the sum of squares of computed contributions to the Dif-estimate under computation by CTGSYL, where the scaling factor RDSCAL (see below) has been factored out. On exit, the corresponding sum of squares updated with the contributions from the current sub-system. If TRANS = 'T' RDSUM is not touched. NOTE: RDSUM only makes sense when CTGSY2 is called by CTGSYL.

RDSCAL Input and output parameter.

RDSCAL is REAL

On entry, scaling factor used to prevent overflow in RDSUM. On exit, RDSCAL is updated w.r.t. the current contributions in RDSUM. If TRANS = 'T', RDSCAL is not touched. NOTE: RDSCAL only makes sense when CTGSY2 is called by CTGSYL.

INFO Output parameter.

INFO is INTEGER

On exit, if INFO is set to =0: Successful exit <0: If INFO = -i, input argument number i is illegal. >0: The matrix pairs (A, D) and (B, E) have common or very close eigenvalues.

Related Information

For this routine in other precisions, please see *dtgsy2*, *stgsy2* and *ztgsy2*.

4.17.165 ctplqt2

ctplqt2 computes a LQ a factorization of a complex "triangular-pentagonal" matrix C, which is composed of a triangular block A and pentagonal block B, using the compact WY representation for Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine ctplqt2(M, N, L, A, LDA, B, LDB, T, LDT, INFO)
```

C specification:

```
#include "armpl.h"

void ctplqt2_(const armpl_int_t *m, const armpl_int_t *n,
              const armpl_int_t *l, armpl_singlecomplex_t *a,
              const armpl_int_t *lda, armpl_singlecomplex_t *b,
              const armpl_int_t *ldb, armpl_singlecomplex_t *t,
              const armpl_int_t *ldt, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The total number of rows of the matrix B. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix B, and the order of the triangular matrix A. N >= 0.

L Input parameter.

L is INTEGER

The number of rows of the lower trapezoidal part of B. MIN(M, N) >= L >= 0. See Further Details.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, M). On entry, the lower triangular M-by-M matrix A. On exit, the elements on and below the diagonal of the array contain the lower triangular matrix L.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, N). On entry, the pentagonal M-by-N matrix B. The first N-L columns are rectangular, and the last L columns are lower trapezoidal. On exit, B contains the pentagonal matrix V. See Further Details.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M).

T Output parameter.

T is COMPLEX

T is an array, dimension (LDT, M). The N-by-N upper triangular factor T of the block reflector. See Further Details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, M)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dtplqt2*, *stplqt2* and *ztplqt2*.

4.17.166 ctpqrt2

ctpqrt2 computes a QR factorization of a complex "triangular-pentagonal" matrix C, which is composed of a triangular block A and pentagonal block B, using the compact WY representation for Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine ctpqrt2(M, N, L, A, LDA, B, LDB, T, LDT, INFO)
```

C specification:

```
#include "armpl.h"

void ctpqrt2_(const armpl_int_t *m, const armpl_int_t *n,
              const armpl_int_t *l, armpl_singlecomplex_t *a,
              const armpl_int_t *lda, armpl_singlecomplex_t *b,
              const armpl_int_t *ldb, armpl_singlecomplex_t *t,
              const armpl_int_t *ldt, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The total number of rows of the matrix B. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix B, and the order of the triangular matrix A. N >= 0.

L Input parameter.

L is INTEGER

The number of rows of the upper trapezoidal part of B. MIN(M, N) >= L >= 0. See Further Details.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the upper triangular N-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the upper triangular matrix R.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, N). On entry, the pentagonal M-by-N matrix B. The first M-L rows are rectangular, and the last L rows are upper trapezoidal. On exit, B contains the pentagonal matrix V. See Further Details.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M).

T Output parameter.

T is COMPLEX

T is an array, dimension (LDT, N). The N-by-N upper triangular factor T of the block reflector. See Further Details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dtpqrt2*, *stpqrt2* and *ztpqrt2*. It also exists with a native C interface as *LAPACKE_ctpqrt2*.

4.17.167 ctprfb

ctprfb applies a complex "triangular-pentagonal" block reflector H or its conjugate transpose H^H to a complex matrix C, which is composed of two blocks A and B, either from the left or right.

Syntax

Fortran specification:

```
use armpl_library

subroutine ctprfb(SIDE, TRANS, DIRECT, STOREV, M, N, K, L, V, LDV, T, LDT, A,
                  LDA, B, LDB, WORK, LDWORK)
```

C specification:

```
#include "armpl.h"

void ctprfb_(const char *side, const char *trans, const char *direct,
             const char *storev, const armpl_int_t *m, const armpl_int_t *n,
```

(continues on next page)

(continued from previous page)

```
                const armpl_int_t *k, const armpl_int_t *l,
                const armpl_singlecomplex_t *v, const armpl_int_t *ldv,
                const armpl_singlecomplex_t *t, const armpl_int_t *ldt,
                armpl_singlecomplex_t *a, const armpl_int_t *lda,
                armpl_singlecomplex_t *b, const armpl_int_t *ldb,
                armpl_singlecomplex_t *work, const armpl_int_t *ldwork, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply H or H^H from the Left = 'R': apply H or H^H from the Right

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply H (No transpose) = 'C': apply H^H (Conjugate transpose)

DIRECT Input parameter.

DIRECT is CHARACTER*1

Indicates how H is formed from a product of elementary reflectors = 'F': H = H(1) H(2) . . . H(k) (Forward) = 'B': H = H(k) . . . H(2) H(1) (Backward)

STOREV Input parameter.

STOREV is CHARACTER*1

Indicates how the vectors which define the elementary reflectors are stored: = 'C': Columns = 'R': Rows

M Input parameter.

M is INTEGER

The number of rows of the matrix B. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix B. N >= 0.

K Input parameter.

K is INTEGER

The order of the matrix T, i.e. the number of elementary reflectors whose product defines the block reflector. K >= 0.

L Input parameter.

L is INTEGER

The order of the trapezoidal part of V. K >= L >= 0. See Further Details.

V Input parameter.

V is COMPLEX

V is an array, dimension. (LDV, K) if STOREV = 'C' (LDV, M) if STOREV = 'R' and SIDE = 'L' (LDV, N) if STOREV = 'R' and SIDE = 'R' The pentagonal matrix V, which contains the elementary reflectors H(1), H(2), . . . , H(K). See Further Details.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. If STOREV = 'C' and SIDE = 'L', LDV >= max(1, M); if STOREV = 'C' and SIDE = 'R', LDV >= max(1, N); if STOREV = 'R', LDV >= K.

T Input parameter.

T is COMPLEX

T is an array, dimension (LDT, K). The triangular K-by-K matrix T in the representation of the block reflector.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= K.

A Input and output parameter.

A is COMPLEX

A is an array, dimension. (LDA, N) if SIDE = 'L' or (LDA, K) if SIDE = 'R' On entry, the K-by-N or M-by-K matrix A. On exit, A is overwritten by the corresponding block of H*C or H^H *C or C*H or C*H^H . See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDC >= max(1, K); If SIDE = 'R', LDC >= max(1, M).

B Input and output parameter.

B is COMPLEX

B is an array, dimension (LDB, N). On entry, the M-by-N matrix B. On exit, B is overwritten by the corresponding block of H*C or H^H *C or C*H or C*H^H . See Further Details.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension. (LDWORK, N) if SIDE = 'L', (LDWORK, K) if SIDE = 'R'.

LDWORK Input parameter.

LDWORK is INTEGER

The leading dimension of the array WORK. If SIDE = 'L', LDWORK >= K; if SIDE = 'R', LDWORK >= M.

Related Information

For this routine in other precisions, please see *dtprfb*, *stprfb* and *ztprfb*. It also exists with a native C interface as *LAPACKE_ctprfb*.

4.17.168 ctpttf

ctpttf copies a triangular matrix A from standard packed format (TP) to rectangular full packed format (TF).

Syntax

Fortran specification:

```
use armpl_library

subroutine ctpttf(TRANSR, UPLO, N, AP, ARF, INFO)
```

C specification:

```
#include "armpl.h"

void ctpttf_(const char *transr, const char *uplo, const armpl_int_t *n,
             const armpl_singlecomplex_t *ap, armpl_singlecomplex_t *arf,
             armpl_int_t *info, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': ARF in Normal format is wanted; = 'C': ARF in Conjugate-transpose format is wanted.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2),. On entry, the upper or lower triangular matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

ARF Output parameter.

ARF is COMPLEX

ARF is an array, dimension (N*(N+1)/2),. On exit, the upper or lower triangular matrix A stored in RFP format. For a further discussion see Notes below.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dtpttf*, *stpttf* and *ztpttf*. It also exists with a native C interface as *LAPACKE_ctpttf*.

4.17.169 ctpttr

ctpttr copies a triangular matrix A from standard packed format (TP) to standard full format (TR).

Syntax

Fortran specification:

```
use armpl_library

subroutine ctpttr(UPLO, N, AP, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void ctpttr_(const char *uplo, const armpl_int_t *n,
             const armpl_singlecomplex_t *ap, armpl_singlecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular. = 'L': A is lower triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2),. On entry, the upper or lower triangular matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

A Output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On exit, the triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dtpttr*, *stpttr* and *ztpttr*. It also exists with a native C interface as *LAPACKE_ctpttr*.

4.17.170 ctrti2

`ctrti2` computes the inverse of a complex upper or lower triangular matrix.

This is the Level 2 BLAS version of the algorithm.

Syntax

Fortran specification:

```
use armpl_library

subroutine ctrti2(UPLO, DIAG, N, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void ctrti2_(const char *uplo, const char *diag, const armpl_int_t *n,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

Specifies whether the matrix `A` is upper or lower triangular. = 'U': Upper triangular = 'L': Lower triangular

DIAG Input parameter.

`DIAG` is `CHARACTER*1`

Specifies whether or not the matrix `A` is unit triangular. = 'N': Non-unit triangular = 'U': Unit triangular

N Input parameter.

`N` is `INTEGER`

The order of the matrix `A`. `N` >= 0.

A Input and output parameter.

`A` is `COMPLEX`

`A` is an array, dimension (`LDA`, `N`). On entry, the triangular matrix `A`. If `UPLO` = 'U', the leading n by n upper triangular part of the array `A` contains the upper triangular matrix, and the strictly lower triangular part of `A` is not referenced. If `UPLO` = 'L', the leading n by n lower triangular part of the array `A` contains the lower triangular matrix, and the strictly upper triangular part of `A` is not referenced. If `DIAG` = 'U', the diagonal elements of `A` are also not referenced and are assumed to be 1.

On exit, the (triangular) inverse of the original matrix, in the same storage format.

LDA Input parameter.

`LDA` is `INTEGER`

The leading dimension of the array `A`. `LDA` >= max(1, `N`).

INFO Output parameter.

`INFO` is `INTEGER`

= 0: successful exit < 0: if `INFO` = -k, the k-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dtrti2*, *strti2* and *ztrti2*.

4.17.171 ctrttf

`ctrttf` copies a triangular matrix A from standard full format (TR) to rectangular full packed format (TF) .

Syntax

Fortran specification:

```
use armpl_library

subroutine ctrttf(TRANSR, UPLO, N, A, LDA, ARF, INFO)
```

C specification:

```
#include "armpl.h"

void ctrttf_(const char *transr, const char *uplo, const armpl_int_t *n,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda,
             armpl_singlecomplex_t *arf, armpl_int_t *info, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': ARF in Normal mode is wanted; = 'C': ARF in Conjugate Transpose mode is wanted;

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the matrix A. LDA >= max(1, N).

ARF Output parameter.

ARF is COMPLEX

ARF is an array, dimension (N*(N+1)/2),. On exit, the upper or lower triangular matrix A stored in RFP format. For a further discussion see Notes below.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dtrttf*, *strttf* and *ztrttf*. It also exists with a native C interface as *LAPACKE_ctrttf*.

4.17.172 ctrttp

ctrttp copies a triangular matrix A from full format (TR) to standard packed format (TP).

Syntax

Fortran specification:

```
use armpl_library

subroutine ctrttp(UPLO, N, A, LDA, AP, INFO)
```

C specification:

```
#include "armpl.h"

void ctrttp_(const char *uplo, const armpl_int_t *n,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda,
             armpl_singlecomplex_t *ap, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

N Input parameter.

N is INTEGER

The order of the matrices AP and A. N >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AP Output parameter.

AP is COMPLEX

AP is an array, dimension (N*(N+1)/2),. On exit, the upper or lower triangular matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dtrttp*, *strttp* and *ztrttp*. It also exists with a native C interface as *LAPACKE_ctrttp*.

4.17.173 cunbdb1

cunbdb1 simultaneously bidiagonalizes the blocks of a tall and skinny matrix X with orthonomal columns:

```
                        [ B11 ]
[ X11 ]   [ P1 |    ] [  0  ]
[-----] = [---------] [-----] Q1**T .
[ X21 ]   [    | P2 ] [ B21 ]
                        [  0  ]
```

X11 is P-by-Q, and X21 is (M-P)-by-Q. Q must be no larger than P, M-P, or M-Q. Routines CUNBDB2, CUNBDB3, and CUNBDB4 handle cases in which Q is not the minimum dimension.

The unitary matrices P1, P2, and Q1 are P-by-P, (M-P)-by-(M-P), and (M-Q)-by-(M-Q), respectively. They are represented implicitly by Householder vectors.

B11 and B12 are Q-by-Q bidiagonal matrices represented implicitly by angles THETA, PHI.

Syntax

Fortran specification:

```
use armpl_library

subroutine cunbdb1(M, P, Q, X11, LDX11, X21, LDX21, THETA, PHI, TAUP1, TAUP2,
                   TAUQ1, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cunbdb1_(const armpl_int_t *m, const armpl_int_t *p,
              const armpl_int_t *q, armpl_singlecomplex_t *x11,
              const armpl_int_t *ldx11, armpl_singlecomplex_t *x21,
              const armpl_int_t *ldx21, float *theta, float *phi,
              armpl_singlecomplex_t *taup1, armpl_singlecomplex_t *taup2,
              armpl_singlecomplex_t *tauq1, armpl_singlecomplex_t *work,
              const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows X11 plus the number of rows in X21.

P Input parameter.

P is INTEGER

The number of rows in X11. 0 <= P <= M.

Q Input parameter.

Q is INTEGER

The number of columns in X11 and X21. 0 <= Q <= MIN(P,M-P,M-Q).

X11 Input and output parameter.

X11 is COMPLEX

X11 is an array, dimension (LDX11, Q). On entry, the top block of the matrix X to be reduced. On exit, the columns of tril(X11) specify reflectors for P1 and the rows of triu(X11,1) specify reflectors for Q1.

LDX11 Input parameter.

LDX11 is INTEGER

The leading dimension of X11. LDX11 >= P.

X21 Input and output parameter.

X21 is COMPLEX

X21 is an array, dimension (LDX21, Q). On entry, the bottom block of the matrix X to be reduced. On exit, the columns of tril(X21) specify reflectors for P2.

LDX21 Input parameter.

LDX21 is INTEGER

The leading dimension of X21. LDX21 >= M-P.

THETA Output parameter.

THETA is REAL

THETA is an array, dimension (Q). The entries of the bidiagonal blocks B11, B21 are defined by THETA and PHI. See Further Details.

PHI Output parameter.

PHI is REAL

PHI is an array, dimension (Q-1). The entries of the bidiagonal blocks B11, B21 are defined by THETA and PHI. See Further Details.

TAUP1 Output parameter.

TAUP1 is COMPLEX

TAUP1 is an array, dimension (P). The scalar factors of the elementary reflectors that define P1.

TAUP2 Output parameter.

TAUP2 is COMPLEX

TAUP2 is an array, dimension (M-P). The scalar factors of the elementary reflectors that define P2.

TAUQ1 Output parameter.

TAUQ1 is COMPLEX

TAUQ1 is an array, dimension (Q). The scalar factors of the elementary reflectors that define Q1.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= M-Q.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *zunbdb1*.

4.17.174 cunbdb2

cunbdb2 simultaneously bidiagonalizes the blocks of a tall and skinny matrix X with orthonomal columns:

```
                        [ B11 ]
[ X11 ]   [ P1 |    ] [  0  ]
[-----] = [---------] [-----] Q1**T .
[ X21 ]   [    | P2 ] [ B21 ]
                        [  0  ]
```

X11 is P-by-Q, and X21 is (M-P)-by-Q. P must be no larger than M-P, Q, or M-Q. Routines CUNBDB1, CUNBDB3, and CUNBDB4 handle cases in which P is not the minimum dimension.

The unitary matrices P1, P2, and Q1 are P-by-P, (M-P)-by-(M-P), and (M-Q)-by-(M-Q), respectively. They are represented implicitly by Householder vectors.

B11 and B12 are P-by-P bidiagonal matrices represented implicitly by angles THETA, PHI.

Syntax

Fortran specification:

```
use armpl_library

subroutine cunbdb2(M, P, Q, X11, LDX11, X21, LDX21, THETA, PHI, TAUP1, TAUP2,
                   TAUQ1, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cunbdb2_(const armpl_int_t *m, const armpl_int_t *p,
              const armpl_int_t *q, armpl_singlecomplex_t *x11,
              const armpl_int_t *ldx11, armpl_singlecomplex_t *x21,
              const armpl_int_t *ldx21, float *theta, float *phi,
              armpl_singlecomplex_t *taup1, armpl_singlecomplex_t *taup2,
              armpl_singlecomplex_t *tauq1, armpl_singlecomplex_t *work,
              const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows X11 plus the number of rows in X21.

P Input parameter.

P is INTEGER

The number of rows in X11. 0 <= P <= min(M-P, Q,M-Q).

Q Input parameter.

Q is INTEGER

The number of columns in X11 and X21. 0 <= Q <= M.

X11 Input and output parameter.

X11 is COMPLEX

X11 is an array, dimension (LDX11, Q). On entry, the top block of the matrix X to be reduced. On exit, the columns of tril(X11) specify reflectors for P1 and the rows of triu(X11,1) specify reflectors for Q1.

LDX11 Input parameter.

LDX11 is INTEGER

The leading dimension of X11. LDX11 >= P.

X21 Input and output parameter.

X21 is COMPLEX

X21 is an array, dimension (LDX21, Q). On entry, the bottom block of the matrix X to be reduced. On exit, the columns of tril(X21) specify reflectors for P2.

LDX21 Input parameter.

LDX21 is INTEGER

The leading dimension of X21. LDX21 >= M-P.

THETA Output parameter.

THETA is REAL

THETA is an array, dimension (Q). The entries of the bidiagonal blocks B11, B21 are defined by THETA and PHI. See Further Details.

PHI Output parameter.

PHI is REAL

PHI is an array, dimension (Q-1). The entries of the bidiagonal blocks B11, B21 are defined by THETA and PHI. See Further Details.

TAUP1 Output parameter.

TAUP1 is COMPLEX

TAUP1 is an array, dimension (P). The scalar factors of the elementary reflectors that define P1.

TAUP2 Output parameter.

TAUP2 is COMPLEX

TAUP2 is an array, dimension (M-P). The scalar factors of the elementary reflectors that define P2.

TAUQ1 Output parameter.

TAUQ1 is COMPLEX

TAUQ1 is an array, dimension (Q). The scalar factors of the elementary reflectors that define Q1.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= M-Q.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *zunbdb2*.

4.17.175 cunbdb3

cunbdb3 simultaneously bidiagonalizes the blocks of a tall and skinny matrix X with orthonomal columns:

```
                         [ B11 ]
[ X11 ]   [ P1 |    ]   [  0  ]
[-----] = [---------]   [-----] Q1**T .
[ X21 ]   [    | P2 ]   [ B21 ]
                         [  0  ]
```

X11 is P-by-Q, and X21 is (M-P)-by-Q. M-P must be no larger than P, Q, or M-Q. Routines CUNBDB1, CUNBDB2, and CUNBDB4 handle cases in which M-P is not the minimum dimension.

The unitary matrices P1, P2, and Q1 are P-by-P, (M-P)-by-(M-P), and (M-Q)-by-(M-Q), respectively. They are represented implicitly by Householder vectors.

B11 and B12 are (M-P)-by-(M-P) bidiagonal matrices represented implicitly by angles THETA, PHI.

Syntax

Fortran specification:

```
use armpl_library

subroutine cunbdb3(M, P, Q, X11, LDX11, X21, LDX21, THETA, PHI, TAUP1, TAUP2,
                   TAUQ1, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cunbdb3_(const armpl_int_t *m, const armpl_int_t *p,
              const armpl_int_t *q, armpl_singlecomplex_t *x11,
              const armpl_int_t *ldx11, armpl_singlecomplex_t *x21,
              const armpl_int_t *ldx21, float *theta, float *phi,
              armpl_singlecomplex_t *taup1, armpl_singlecomplex_t *taup2,
              armpl_singlecomplex_t *tauq1, armpl_singlecomplex_t *work,
              const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows X11 plus the number of rows in X21.

P Input parameter.

P is INTEGER

The number of rows in X11. 0 <= P <= M. M-P <= min(P, Q,M-Q).

Q Input parameter.

Q is INTEGER

The number of columns in X11 and X21. 0 <= Q <= M.

X11 Input and output parameter.

X11 is COMPLEX

X11 is an array, dimension (LDX11, Q). On entry, the top block of the matrix X to be reduced. On exit, the columns of tril(X11) specify reflectors for P1 and the rows of triu(X11,1) specify reflectors for Q1.

LDX11 Input parameter.

LDX11 is INTEGER

The leading dimension of X11. LDX11 >= P.

X21 Input and output parameter.

X21 is COMPLEX

X21 is an array, dimension (LDX21, Q). On entry, the bottom block of the matrix X to be reduced. On exit, the columns of tril(X21) specify reflectors for P2.

LDX21 Input parameter.

LDX21 is INTEGER

The leading dimension of X21. LDX21 >= M-P.

THETA Output parameter.

THETA is REAL

THETA is an array, dimension (Q). The entries of the bidiagonal blocks B11, B21 are defined by THETA and PHI. See Further Details.

PHI Output parameter.

PHI is REAL

PHI is an array, dimension (Q-1). The entries of the bidiagonal blocks B11, B21 are defined by THETA and PHI. See Further Details.

TAUP1 Output parameter.

TAUP1 is COMPLEX

TAUP1 is an array, dimension (P). The scalar factors of the elementary reflectors that define P1.

TAUP2 Output parameter.

TAUP2 is COMPLEX

TAUP2 is an array, dimension (M-P). The scalar factors of the elementary reflectors that define P2.

TAUQ1 Output parameter.

TAUQ1 is COMPLEX

TAUQ1 is an array, dimension (Q). The scalar factors of the elementary reflectors that define Q1.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= M-Q.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *zunbdb3*.

4.17.176 cunbdb4

cunbdb4 simultaneously bidiagonalizes the blocks of a tall and skinny matrix X with orthonomal columns:

```
                        [ B11 ]
[ X11 ]   [ P1 |    ] [  0  ]
[-----] = [---------] [-----] Q1**T .
[ X21 ]   [    | P2 ] [ B21 ]
                        [  0  ]
```

X11 is P-by-Q, and X21 is (M-P)-by-Q. M-Q must be no larger than P, M-P, or Q. Routines CUNBDB1, CUNBDB2, and CUNBDB3 handle cases in which M-Q is not the minimum dimension.

The unitary matrices P1, P2, and Q1 are P-by-P, (M-P)-by-(M-P), and (M-Q)-by-(M-Q), respectively. They are represented implicitly by Householder vectors.

B11 and B12 are (M-Q)-by-(M-Q) bidiagonal matrices represented implicitly by angles THETA, PHI.

Syntax

Fortran specification:

```
use armpl_library

subroutine cunbdb4(M, P, Q, X11, LDX11, X21, LDX21, THETA, PHI, TAUP1, TAUP2,
                   TAUQ1, PHANTOM, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cunbdb4_(const armpl_int_t *m, const armpl_int_t *p,
              const armpl_int_t *q, armpl_singlecomplex_t *x11,
              const armpl_int_t *ldx11, armpl_singlecomplex_t *x21,
              const armpl_int_t *ldx21, float *theta, float *phi,
              armpl_singlecomplex_t *taup1, armpl_singlecomplex_t *taup2,
              armpl_singlecomplex_t *tauq1, armpl_singlecomplex_t *phantom,
              armpl_singlecomplex_t *work, const armpl_int_t *lwork,
              armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows X11 plus the number of rows in X21.

P Input parameter.

P is INTEGER

The number of rows in X11. 0 <= P <= M.

Q Input parameter.

Q is INTEGER

The number of columns in X11 and X21. 0 <= Q <= M and M-Q <= min(P,M-P, Q).

X11 Input and output parameter.

X11 is COMPLEX

X11 is an array, dimension (LDX11, Q). On entry, the top block of the matrix X to be reduced. On exit, the columns of tril(X11) specify reflectors for P1 and the rows of triu(X11,1) specify reflectors for Q1.

LDX11 Input parameter.

LDX11 is INTEGER

The leading dimension of X11. LDX11 >= P.

X21 Input and output parameter.

X21 is COMPLEX

X21 is an array, dimension (LDX21, Q). On entry, the bottom block of the matrix X to be reduced. On exit, the columns of tril(X21) specify reflectors for P2.

LDX21 Input parameter.

LDX21 is INTEGER

The leading dimension of X21. LDX21 >= M-P.

THETA Output parameter.

THETA is REAL

THETA is an array, dimension (Q). The entries of the bidiagonal blocks B11, B21 are defined by THETA and PHI. See Further Details.

PHI Output parameter.

PHI is REAL

PHI is an array, dimension (Q-1). The entries of the bidiagonal blocks B11, B21 are defined by THETA and PHI. See Further Details.

TAUP1 Output parameter.

TAUP1 is COMPLEX

TAUP1 is an array, dimension (P). The scalar factors of the elementary reflectors that define P1.

TAUP2 Output parameter.

TAUP2 is COMPLEX

TAUP2 is an array, dimension (M-P). The scalar factors of the elementary reflectors that define P2.

TAUQ1 Output parameter.

TAUQ1 is COMPLEX

TAUQ1 is an array, dimension (Q). The scalar factors of the elementary reflectors that define Q1.

PHANTOM Output parameter.

PHANTOM is COMPLEX

PHANTOM is an array, dimension (M). The routine computes an M-by-1 column vector Y that is orthogonal to the columns of [X11; X21]. PHANTOM(1:P) and PHANTOM(P+1:M) contain Householder vectors for Y(1:P) and Y(P+1:M), respectively.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= M-Q.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *zunbdb4*.

4.17.177 cunbdb5

`cunbdb5` orthogonalizes the column vector

```
X = [ X1 ]
    [ X2 ]
```

with respect to the columns of

```
Q = [ Q1 ] .
    [ Q2 ]
```

The columns of Q must be orthonormal.

If the projection is zero according to Kahan's "twice is enough" criterion, then some other vector from the orthogonal complement is returned. This vector is chosen in an arbitrary but deterministic way.

Syntax

Fortran specification:

```
use armpl_library

subroutine cunbdb5(M1, M2, N, X1, INCX1, X2, INCX2, Q1, LDQ1, Q2, LDQ2, WORK,
                   LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cunbdb5_(const armpl_int_t *m1, const armpl_int_t *m2,
              const armpl_int_t *n, armpl_singlecomplex_t *x1,
              const armpl_int_t *incx1, armpl_singlecomplex_t *x2,
              const armpl_int_t *incx2, armpl_singlecomplex_t *q1,
              const armpl_int_t *ldq1, armpl_singlecomplex_t *q2,
              const armpl_int_t *ldq2, armpl_singlecomplex_t *work,
              const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M1 Input parameter.

`M1` is `INTEGER`

The dimension of `X1` and the number of rows in `Q1`. 0 <= `M1`.

M2 Input parameter.

`M2` is `INTEGER`

The dimension of `X2` and the number of rows in `Q2`. 0 <= `M2`.

N Input parameter.

`N` is `INTEGER`

The number of columns in `Q1` and `Q2`. 0 <= `N`.

X1 Input and output parameter.

X1 is COMPLEX

X1 is an array, dimension (M1). On entry, the top part of the vector to be orthogonalized. On exit, the top part of the projected vector.

INCX1 Input parameter.

INCX1 is INTEGER

Increment for entries of X1.

X2 Input and output parameter.

X2 is COMPLEX

X2 is an array, dimension (M2). On entry, the bottom part of the vector to be orthogonalized. On exit, the bottom part of the projected vector.

INCX2 Input parameter.

INCX2 is INTEGER

Increment for entries of X2.

Q1 Input parameter.

Q1 is COMPLEX

Q1 is an array, dimension (LDQ1, N). The top part of the orthonormal basis matrix.

LDQ1 Input parameter.

LDQ1 is INTEGER

The leading dimension of Q1. LDQ1 >= M1.

Q2 Input parameter.

Q2 is COMPLEX

Q2 is an array, dimension (LDQ2, N). The bottom part of the orthonormal basis matrix.

LDQ2 Input parameter.

LDQ2 is INTEGER

The leading dimension of Q2. LDQ2 >= M2.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= N.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *zunbdb5*.

4.17.178 cunbdb6

`cunbdb6` orthogonalizes the column vector

```
X = [ X1 ]
    [ X2 ]
```

with respect to the columns of

```
Q = [ Q1 ] .
    [ Q2 ]
```

The columns of Q must be orthonormal.

If the projection is zero according to Kahan's "twice is enough" criterion, then the zero vector is returned.

Syntax

Fortran specification:

```
use armpl_library

subroutine cunbdb6(M1, M2, N, X1, INCX1, X2, INCX2, Q1, LDQ1, Q2, LDQ2, WORK,
                   LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void cunbdb6_(const armpl_int_t *m1, const armpl_int_t *m2,
              const armpl_int_t *n, armpl_singlecomplex_t *x1,
              const armpl_int_t *incx1, armpl_singlecomplex_t *x2,
              const armpl_int_t *incx2, armpl_singlecomplex_t *q1,
              const armpl_int_t *ldq1, armpl_singlecomplex_t *q2,
              const armpl_int_t *ldq2, armpl_singlecomplex_t *work,
              const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M1 Input parameter.

`M1` is `INTEGER`

The dimension of `X1` and the number of rows in `Q1`. 0 <= `M1`.

M2 Input parameter.

`M2` is `INTEGER`

The dimension of `X2` and the number of rows in `Q2`. 0 <= `M2`.

N Input parameter.

`N` is `INTEGER`

The number of columns in `Q1` and `Q2`. 0 <= `N`.

X1 Input and output parameter.

`X1` is `COMPLEX`

`X1` is an array, dimension (`M1`). On entry, the top part of the vector to be orthogonalized. On exit, the top part of the projected vector.

INCX1 Input parameter.

INCX1 is INTEGER

Increment for entries of X1.

X2 Input and output parameter.

X2 is COMPLEX

X2 is an array, dimension (M2). On entry, the bottom part of the vector to be orthogonalized. On exit, the bottom part of the projected vector.

INCX2 Input parameter.

INCX2 is INTEGER

Increment for entries of X2.

Q1 Input parameter.

Q1 is COMPLEX

Q1 is an array, dimension (LDQ1, N). The top part of the orthonormal basis matrix.

LDQ1 Input parameter.

LDQ1 is INTEGER

The leading dimension of Q1. LDQ1 >= M1.

Q2 Input parameter.

Q2 is COMPLEX

Q2 is an array, dimension (LDQ2, N). The bottom part of the orthonormal basis matrix.

LDQ2 Input parameter.

LDQ2 is INTEGER

The leading dimension of Q2. LDQ2 >= M2.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= N.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *zunbdb6*.

4.17.179 cung2l

cung2l generates an m by n complex matrix Q with orthonormal columns, which is defined as the last n columns of a product of k elementary reflectors of order m

```
Q  =  H(k) . . . H(2) H(1)
```

as returned by CGEQLF.

Syntax

Fortran specification:

```
use armpl_library

subroutine cung2l(M, N, K, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void cung2l_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *k,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const armpl_singlecomplex_t *tau, armpl_singlecomplex_t *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix Q. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix Q. M >= N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. N >= K >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the (n-k+i)-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by CGEQLF in the last k columns of its array argument A. On exit, the m-by-n matrix Q.

LDA Input parameter.

LDA is INTEGER

The first dimension of the array A. LDA >= max(1, M).

TAU Input parameter.

TAU is COMPLEX

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by CGEQLF.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument has an illegal value

Related Information

For this routine in other precisions, please see *zung2l*.

4.17.180 cung2r

cung2r generates an m by n complex matrix Q with orthonormal columns, which is defined as the first n columns of a product of k elementary reflectors of order m

```
Q  =  H(1) H(2) . . . H(k)
```

as returned by CGEQRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine cung2r(M, N, K, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void cung2r_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *k,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const armpl_singlecomplex_t *tau, armpl_singlecomplex_t *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix Q. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix Q. M >= N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. N >= K >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the i-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,. . . ,k, as returned by CGEQRF in the first k columns of its array argument A. On exit, the m by n matrix Q.

LDA Input parameter.

LDA is INTEGER

The first dimension of the array A. LDA >= max(1, M).

TAU Input parameter.

TAU is COMPLEX

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by CGEQRF.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument has an illegal value

Related Information

For this routine in other precisions, please see *zung2r*.

4.17.181 cungl2

cungl2 generates an m-by-n complex matrix Q with orthonormal rows, which is defined as the first m rows of a product of k elementary reflectors of order n

```
Q  =  H(k)**H . . . H(2)**H H(1)**H
```

as returned by CGELQF.

Syntax

Fortran specification:

```
use armpl_library

subroutine cungl2(M, N, K, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void cungl2_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *k,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const armpl_singlecomplex_t *tau, armpl_singlecomplex_t *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix Q. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix Q. N >= M.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. M >= K >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,. . . ,k, as returned by CGELQF in the first k rows of its array argument A. On exit, the m by n matrix Q.

LDA Input parameter.

LDA is INTEGER

The first dimension of the array A. LDA >= max(1, M).

TAU Input parameter.

TAU is COMPLEX

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by CGELQF.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (M) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument has an illegal value

Related Information

For this routine in other precisions, please see *zungl2*.

4.17.182 cungr2

cungr2 generates an m by n complex matrix Q with orthonormal rows, which is defined as the last m rows of a product of k elementary reflectors of order n

```
Q  =  H(1)**H H(2)**H . . . H(k)**H
```

as returned by CGERQF.

Syntax

Fortran specification:

```
use armpl_library

subroutine cungr2(M, N, K, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void cungr2_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *k,
             armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const armpl_singlecomplex_t *tau, armpl_singlecomplex_t *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix Q. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix Q. N >= M.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. M >= K >= 0.

A Input and output parameter.

A is COMPLEX

A is an array, dimension (LDA, N). On entry, the (m-k+i)-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by CGERQF in the last k rows of its array argument A. On exit, the m-by-n matrix Q.

LDA Input parameter.

LDA is INTEGER

The first dimension of the array A. LDA >= max(1, M).

TAU Input parameter.

TAU is COMPLEX

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by CGERQF.

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension (M) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument has an illegal value

Related Information

For this routine in other precisions, please see *zungr2*.

4.17.183 cunm2l

cunm2l overwrites the general complex m-by-n matrix C with

```
Q * C  if SIDE = 'L' and TRANS = 'N', or

Q**H* C  if SIDE = 'L' and TRANS = 'C', or

C * Q  if SIDE = 'R' and TRANS = 'N', or

C * Q**H if SIDE = 'R' and TRANS = 'C',
```

where Q is a complex unitary matrix defined as the product of k elementary reflectors

```
Q = H(k) . . . H(2) H(1)
```

as returned by CGEQLF. Q is of order m if SIDE = 'L' and of order n if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine cunm2l(SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void cunm2l_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const armpl_singlecomplex_t *tau, armpl_singlecomplex_t *c,
             const armpl_int_t *ldc, armpl_singlecomplex_t *work,
             armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left = 'R': apply Q or Q^H from the Right

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply Q (No transpose) = 'C': apply Q^H (Conjugate transpose)

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, K). The i-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by CGEQLF in the last k columns of its array argument A. A is modified by the routine but restored on exit.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDA >= max(1, M); if SIDE = 'R', LDA >= max(1, N).

TAU Input parameter.

TAU is COMPLEX

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by CGEQLF.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). On entry, the m-by-n matrix C. On exit, C is overwritten by Q* C or Q^H * C or C*Q^H or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension. (N) if SIDE = 'L', (M) if SIDE = 'R'

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zunm2l*.

4.17.184 cunm2r

cunm2r overwrites the general complex m-by-n matrix C with

```
Q * C  if SIDE = 'L' and TRANS = 'N', or

Q**H* C  if SIDE = 'L' and TRANS = 'C', or

C * Q  if SIDE = 'R' and TRANS = 'N', or

C * Q**H if SIDE = 'R' and TRANS = 'C',
```

where Q is a complex unitary matrix defined as the product of k elementary reflectors

```
Q = H(1) H(2) . . . H(k)
```

as returned by CGEQRF. Q is of order m if SIDE = 'L' and of order n if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine cunm2r(SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void cunm2r_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const armpl_singlecomplex_t *tau, armpl_singlecomplex_t *c,
             const armpl_int_t *ldc, armpl_singlecomplex_t *work,
             armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left = 'R': apply Q or Q^H from the Right

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply Q (No transpose) = 'C': apply Q^H (Conjugate transpose)

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension (LDA, K). The i-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,. . . ,k, as returned by CGEQRF in the first k columns of its array argument A. A is modified by the routine but restored on exit.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDA >= max(1, M); if SIDE = 'R', LDA >= max(1, N).

TAU Input parameter.

TAU is COMPLEX

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by CGEQRF.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). On entry, the m-by-n matrix C. On exit, C is overwritten by Q* C or Q^H * C or C*Q^H or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension. (N) if SIDE = 'L', (M) if SIDE = 'R'

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zunm2r*.

4.17.185 cunml2

cunml2 overwrites the general complex m-by-n matrix C with

```
Q * C  if SIDE = 'L' and TRANS = 'N', or

Q**H* C  if SIDE = 'L' and TRANS = 'C', or

C * Q  if SIDE = 'R' and TRANS = 'N', or

C * Q**H if SIDE = 'R' and TRANS = 'C',
```

where Q is a complex unitary matrix defined as the product of k elementary reflectors

```
Q = H(k)**H . . . H(2)**H H(1)**H
```

as returned by CGELQF. Q is of order m if SIDE = 'L' and of order n if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine cunml2(SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void cunml2_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const armpl_singlecomplex_t *tau, armpl_singlecomplex_t *c,
             const armpl_int_t *ldc, armpl_singlecomplex_t *work,
             armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left = 'R': apply Q or Q^H from the Right

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply Q (No transpose) = 'C': apply Q^H (Conjugate transpose)

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension. (LDA, M) if SIDE = 'L', (LDA, N) if SIDE = 'R' The i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by CGELQF in the first k rows of its array argument A. A is modified by the routine but restored on exit.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, K).

TAU Input parameter.

TAU is COMPLEX

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by CGELQF.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). On entry, the m-by-n matrix C. On exit, C is overwritten by Q* C or Q^H * C or C*Q^H or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension. (N) if SIDE = 'L', (M) if SIDE = 'R'

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zunml2*.

4.17.186 cunmr2

cunmr2 overwrites the general complex m-by-n matrix C with

```
Q * C  if SIDE = 'L' and TRANS = 'N', or

Q**H* C  if SIDE = 'L' and TRANS = 'C', or

C * Q  if SIDE = 'R' and TRANS = 'N', or

C * Q**H if SIDE = 'R' and TRANS = 'C',
```

where Q is a complex unitary matrix defined as the product of k elementary reflectors

```
Q = H(1)**H H(2)**H . . . H(k)**H
```

as returned by CGERQF. Q is of order m if SIDE = 'L' and of order n if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine cunmr2(SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void cunmr2_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const armpl_singlecomplex_t *tau, armpl_singlecomplex_t *c,
```

(continues on next page)

(continued from previous page)

```
            const armpl_int_t *ldc, armpl_singlecomplex_t *work,
            armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left = 'R': apply Q or Q^H from the Right

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply Q (No transpose) = 'C': apply Q^H (Conjugate transpose)

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension. (LDA, M) if SIDE = 'L', (LDA, N) if SIDE = 'R' The i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by CGERQF in the last k rows of its array argument A. A is modified by the routine but restored on exit.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, K).

TAU Input parameter.

TAU is COMPLEX

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by CGERQF.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). On entry, the m-by-n matrix C. On exit, C is overwritten by Q* C or Q^H * C or C*Q^H or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension. (N) if SIDE = 'L', (M) if SIDE = 'R'

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zunmr2*.

4.17.187 cunmr3

cunmr3 overwrites the general complex m by n matrix C with

```
Q * C  if SIDE = 'L' and TRANS = 'N', or

Q**H* C  if SIDE = 'L' and TRANS = 'C', or

C * Q  if SIDE = 'R' and TRANS = 'N', or

C * Q**H if SIDE = 'R' and TRANS = 'C',
```

where Q is a complex unitary matrix defined as the product of k elementary reflectors

```
Q = H(1) H(2) . . . H(k)
```

as returned by CTZRZF. Q is of order m if SIDE = 'L' and of order n if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine cunmr3(SIDE, TRANS, M, N, K, L, A, LDA, TAU, C, LDC, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void cunmr3_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k, const armpl_int_t *l,
             const armpl_singlecomplex_t *a, const armpl_int_t *lda,
             const armpl_singlecomplex_t *tau, armpl_singlecomplex_t *c,
             const armpl_int_t *ldc, armpl_singlecomplex_t *work,
             armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left = 'R': apply Q or Q^H from the Right

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply Q (No transpose) = 'C': apply Q^H (Conjugate transpose)

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

L Input parameter.

L is INTEGER

The number of columns of the matrix A containing the meaningful part of the Householder reflectors. If SIDE = 'L', M >= L >= 0, if SIDE = 'R', N >= L >= 0.

A Input parameter.

A is COMPLEX

A is an array, dimension. (LDA, M) if SIDE = 'L', (LDA, N) if SIDE = 'R' The i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by CTZRZF in the last k rows of its array argument A. A is modified by the routine but restored on exit.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, K).

TAU Input parameter.

TAU is COMPLEX

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by CTZRZF.

C Input and output parameter.

C is COMPLEX

C is an array, dimension (LDC, N). On entry, the m-by-n matrix C. On exit, C is overwritten by Q* C or Q^H * C or C*Q^H or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX

WORK is an array, dimension. (N) if SIDE = 'L', (M) if SIDE = 'R'

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *zunmr3*.

4.17.188 dgbtf2

dgbtf2 computes an LU factorization of a real m-by-n band matrix A using partial pivoting with row interchanges. This is the unblocked version of the algorithm, calling Level 2 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgbtf2(M, N, KL, KU, AB, LDAB, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void dgbtf2_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *kl, const armpl_int_t *ku, double *ab,
             const armpl_int_t *ldab, armpl_int_t *ipiv, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

AB Input and output parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows KL+1 to 2*KL+KU+1; rows 1 to KL of the array need not be set. The j-th column of A is stored in the j-th column of the array AB as follows: AB(kl+ku+1+i-j,j) = A(i,j) for max(1,j-ku)<=i<=min(m,j+kl)

On exit, details of the factorization: U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1. See below for further details.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= 2*KL+KU+1.

IPIV Output parameter.

IPIV is INTEGER array, dimension (min(M, N))

The pivot indices; for 1 <= i <= min(M, N), row i of the matrix was interchanged with row IPIV(i).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = +i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *cgbtf2*, *sgbtf2* and *zgbtf2*.

4.17.189 dgebd2

dgebd2 reduces a real general m by n matrix A to upper or lower bidiagonal form B by an orthogonal transformation: $Q^T * A * P = B$.

If m >= n, B is upper bidiagonal; if m < n, B is lower bidiagonal.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgebd2(M, N, A, LDA, D, E, TAUQ, TAUP, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgebd2_(const armpl_int_t *m, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, double *d, double *e, double *tauq,
             double *taup, double *work, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows in the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns in the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the m by n general matrix to be reduced. On exit, if m >= n, the diagonal and the first superdiagonal are overwritten with the upper bidiagonal matrix B; the elements below the diagonal, with the array TAUQ, represent the orthogonal matrix Q as a product of elementary reflectors, and the elements above the first superdiagonal, with the array TAUP, represent the orthogonal matrix P as a product of elementary reflectors; if m < n, the diagonal and the first subdiagonal are overwritten with the lower bidiagonal matrix B; the elements below the first subdiagonal, with the array TAUQ, represent the orthogonal matrix Q as a product of elementary reflectors, and the elements above the diagonal, with the array TAUP, represent the orthogonal matrix P as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

D Output parameter.

D is DOUBLE PRECISION

D is an array, dimension (min(M, N)). The diagonal elements of the bidiagonal matrix B: D(i) = A(i,i).

E Output parameter.

E is DOUBLE PRECISION

E is an array, dimension (min(M, N)-1). The off-diagonal elements of the bidiagonal matrix B: if m >= n, E(i) = A(i,i+1) for i = 1,2,...,n-1; if m < n, E(i) = A(i+1,i) for i = 1,2,...,m-1.

TAUQ Output parameter.

TAUQ is DOUBLE PRECISION

TAUQ is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors which represent the orthogonal matrix Q. See Further Details.

TAUP Output parameter.

TAUP is DOUBLE PRECISION

TAUP is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors which represent the orthogonal matrix P. See Further Details.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (max(M, N)) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cgebd2*, *sgebd2* and *zgebd2*.

4.17.190 dgehd2

dgehd2 reduces a real general matrix A to upper Hessenberg form H by an orthogonal similarity transformation: $Q^T * A * Q = H$.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgehd2(N, ILO, IHI, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgehd2_(const armpl_int_t *n, const armpl_int_t *ilo,
             const armpl_int_t *ihi, double *a, const armpl_int_t *lda,
             double *tau, double *work, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

It is assumed that A is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N. ILO and IHI are normally set by a previous call to DGEBAL; otherwise they should be set to 1 and N respectively. See Further Details. 1 <= ILO <= IHI <= max(1, N).

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the n by n general matrix to be reduced. On exit, the upper triangle and the first subdiagonal of A are overwritten with the upper Hessenberg matrix H, and the elements below the first subdiagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TAU Output parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (N-1). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cgehd2*, *sgehd2* and *zgehd2*.

4.17.191 dgelq2

dgelq2 computes an LQ factorization of a real m by n matrix A: A = L * Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgelq2(M, N, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgelq2_(const armpl_int_t *m, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, double *tau, double *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the m by n matrix A. On exit, the elements on and below the diagonal of the array contain the m by min(m,n) lower trapezoidal matrix L (L is lower triangular if m <= n); the elements above the diagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (M) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgelq2*, *sgelq2* and *zgelq2*. It also exists with a native C interface as *LAPACKE_dgelq2*.

4.17.192 dgelqt3

dgelqt3 recursively computes a LQ factorization of a real M-by-N matrix A, using the compact WY representation of Q.

Based on the algorithm of Elmroth and Gustavson, IBM J. Res. Develop. Vol 44 No. 4 July 2000.

Syntax

Fortran specification:

```
use armpl_library

recursive subroutine dgelqt3(M, N, A, LDA, T, LDT, INFO)
```

C specification:

```
#include "armpl.h"

void dgelqt3_(const armpl_int_t *m, const armpl_int_t *n, double *a,
              const armpl_int_t *lda, double *t, const armpl_int_t *ldt,
              armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M =< N.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the real M-by-N matrix A. On exit, the elements on and below the diagonal contain the N-by-N lower triangular matrix L; the elements above the diagonal are the rows of V. See below for further details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

T Output parameter.

T is DOUBLE PRECISION

T is an array, dimension (LDT, N). The N-by-N upper triangular factor of the block reflector. The elements on and above the diagonal contain the block reflector T; the elements below the diagonal are not used. See below for further details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgelqt3*, *sgelqt3* and *zgelqt3*.

4.17.193 dgeql2

dgeql2 computes a QL factorization of a real m by n matrix A: A = Q * L.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgeql2(M, N, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgeql2_(const armpl_int_t *m, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, double *tau, double *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the m by n matrix A. On exit, if m >= n, the lower triangle of the subarray A(m-n+1:m,1:n) contains the n by n lower triangular matrix L; if m <= n, the elements on and below the (n-m)-th superdiagonal contain the m by n lower trapezoidal matrix L; the remaining elements, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgeql2*, *sgeql2* and *zgeql2*.

4.17.194 dgeqr2

dgeqr2 computes a QR factorization of a real m by n matrix A: A = Q * R.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgeqr2(M, N, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgeqr2_(const armpl_int_t *m, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, double *tau, double *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the m by n matrix A. On exit, the elements on and above the diagonal of the array contain the min(m,n) by n upper trapezoidal matrix R (R is upper triangular if m >= n); the elements below the diagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgeqr2*, *sgeqr2* and *zgeqr2*. It also exists with a native C interface as *LAPACKE_dgeqr2*.

4.17.195 dgeqr2p

dgeqr2p computes a QR factorization of a real m by n matrix A: A = Q * R. The diagonal entries of R are nonnegative.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgeqr2p(M, N, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgeqr2p_(const armpl_int_t *m, const armpl_int_t *n, double *a,
              const armpl_int_t *lda, double *tau, double *work,
              armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the m by n matrix A. On exit, the elements on and above the diagonal of the array contain the min(m,n) by n upper trapezoidal matrix R (R is upper triangular if m >= n). The diagonal entries of R are nonnegative; the elements below the diagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgeqr2p*, *sgeqr2p* and *zgeqr2p*.

4.17.196 dgeqrt2

dgeqrt2 computes a QR factorization of a real M-by-N matrix A, using the compact WY representation of Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgeqrt2(M, N, A, LDA, T, LDT, INFO)
```

C specification:

```
#include "armpl.h"

void dgeqrt2_(const armpl_int_t *m, const armpl_int_t *n, double *a,
              const armpl_int_t *lda, double *t, const armpl_int_t *ldt,
              armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= N.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the real M-by-N matrix A. On exit, the elements on and above the diagonal contain the N-by-N upper triangular matrix R; the elements below the diagonal are the columns of V. See below for further details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

T Output parameter.

T is DOUBLE PRECISION

T is an array, dimension (LDT, N). The N-by-N upper triangular factor of the block reflector. The elements on and above the diagonal contain the block reflector T; the elements below the diagonal are not used. See below for further details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgeqrt2*, *sgeqrt2* and *zgeqrt2*. It also exists with a native C interface as *LAPACKE_dgeqrt2*.

4.17.197 dgeqrt3

dgeqrt3 recursively computes a QR factorization of a real M-by-N matrix A, using the compact WY representation of Q.

Based on the algorithm of Elmroth and Gustavson, IBM J. Res. Develop. Vol 44 No. 4 July 2000.

Syntax

Fortran specification:

```
use armpl_library

recursive subroutine dgeqrt3(M, N, A, LDA, T, LDT, INFO)
```

C specification:

```
#include "armpl.h"

void dgeqrt3_(const armpl_int_t *m, const armpl_int_t *n, double *a,
              const armpl_int_t *lda, double *t, const armpl_int_t *ldt,
              armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= N.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the real M-by-N matrix A. On exit, the elements on and above the diagonal contain the N-by-N upper triangular matrix R; the elements below the diagonal are the columns of V. See below for further details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

T Output parameter.

T is DOUBLE PRECISION

T is an array, dimension (LDT, N). The N-by-N upper triangular factor of the block reflector. The elements on and above the diagonal contain the block reflector T; the elements below the diagonal are not used. See below for further details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgeqrt3*, *sgeqrt3* and *zgeqrt3*. It also exists with a native C interface as *LAPACKE_dgeqrt3*.

4.17.198 dgerq2

`dgerq2` computes an RQ factorization of a real m by n matrix A: A = R * Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgerq2(M, N, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgerq2_(const armpl_int_t *m, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, double *tau, double *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

`M` is `INTEGER`

The number of rows of the matrix `A`. `M` >= 0.

N Input parameter.

`N` is `INTEGER`

The number of columns of the matrix `A`. `N` >= 0.

A Input and output parameter.

`A` is `DOUBLE PRECISION`

`A` is an array, dimension (`LDA`, `N`). On entry, the m by n matrix `A`. On exit, if m <= n, the upper triangle of the subarray A(1:m,n-m+1:n) contains the m by m upper triangular matrix R; if m >= n, the elements on and above the (m-n)-th subdiagonal contain the m by n upper trapezoidal matrix R; the remaining elements, with the array `TAU`, represent the orthogonal matrix Q as a product of elementary reflectors (see Further Details).

LDA Input parameter.

`LDA` is `INTEGER`

The leading dimension of the array `A`. `LDA` >= max(1, `M`).

TAU Output parameter.

`TAU` is `DOUBLE PRECISION`

`TAU` is an array, dimension (min(`M`, `N`)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

`WORK` is `DOUBLE PRECISION`

WORK is an array, dimension (M) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgerq2*, *sgerq2* and *zgerq2*.

4.17.199 dgesc2

dgesc2 solves a system of linear equations

```
A * X = scale* RHS
```

with a general N-by-N matrix A using the LU factorization with complete pivoting computed by DGETC2.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgesc2(N, A, LDA, RHS, IPIV, JPIV, SCALE)
```

C specification:

```
#include "armpl.h"

void dgesc2_(const armpl_int_t *n, const double *a, const armpl_int_t *lda,
             double *rhs, const armpl_int_t *ipiv, const armpl_int_t *jpiv,
             double *scale);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the LU part of the factorization of the n-by-n matrix A computed by DGETC2: A = P * L * U * Q

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

RHS Input and output parameter.

RHS is DOUBLE PRECISION

RHS is an array, dimension (N).. On entry, the right hand side vector b. On exit, the solution vector X.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N).

The pivot indices; for 1 <= i <= N, row i of the matrix has been interchanged with row IPIV(i).

JPIV Input parameter.

JPIV is INTEGER array, dimension (N).

The pivot indices; for 1 <= j <= N, column j of the matrix has been interchanged with column JPIV(j).

SCALE Output parameter.

SCALE is DOUBLE PRECISION

On exit, SCALE contains the scale factor. SCALE is chosen 0 <= SCALE <= 1 to prevent owerflow in the solution.

Related Information

For this routine in other precisions, please see *cgesc2*, *sgesc2* and *zgesc2*.

4.17.200 dgetc2

dgetc2 computes an LU factorization with complete pivoting of the n-by-n matrix A. The factorization has the form A = P * L * U * Q, where P and Q are permutation matrices, L is lower triangular with unit diagonal elements and U is upper triangular.

This is the Level 2 BLAS algorithm.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgetc2(N, A, LDA, IPIV, JPIV, INFO)
```

C specification:

```
#include "armpl.h"

void dgetc2_(const armpl_int_t *n, double *a, const armpl_int_t *lda,
             armpl_int_t *ipiv, armpl_int_t *jpiv, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the n-by-n matrix A to be factored. On exit, the factors L and U from the factorization A = P*L*U*Q; the unit diagonal elements of L are not stored. If U(k, k) appears to be less than SMIN, U(k, k) is given the value of SMIN, i.e., giving a nonsingular perturbed system.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension(N).

The pivot indices; for 1 <= i <= N, row i of the matrix has been interchanged with row IPIV(i).

JPIV Output parameter.

JPIV is INTEGER array, dimension(N).

The pivot indices; for 1 <= j <= N, column j of the matrix has been interchanged with column JPIV(j).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit > 0: if INFO = k, U(k, k) is likely to produce owerflow if we try to solve for x in Ax = b. So U is perturbed to avoid the overflow.

Related Information

For this routine in other precisions, please see *cgetc2*, *sgetc2* and *zgetc2*.

4.17.201 dgetf2

dgetf2 computes an LU factorization of a general m-by-n matrix A using partial pivoting with row interchanges.

The factorization has the form

```
A = P * L * U
```

where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if m > n), and U is upper triangular (upper trapezoidal if m < n).

This is the right-looking Level 2 BLAS version of the algorithm.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgetf2(M, N, A, LDA, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void dgetf2_(const armpl_int_t *m, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, armpl_int_t *ipiv, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the m by n matrix to be factored. On exit, the factors L and U from the factorization A = P*L*U; the unit diagonal elements of L are not stored.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

IPIV Output parameter.

IPIV is INTEGER array, dimension (min(M, N))

The pivot indices; for 1 <= i <= min(M, N), row i of the matrix was interchanged with row IPIV(i).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value > 0: if INFO = k, U(k,k) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *cgetf2*, *sgetf2* and *zgetf2*. It also exists with a native C interface as *LAPACKE_dgetf2*.

4.17.202 dgsvj0

dgsvj0 is called from DGESVJ as a pre-processor and that is its main purpose. It applies Jacobi rotations in the same way as DGESVJ does, but it does not check convergence (stopping criterion). Few tuning parameters (marked by [TP]) are available for the implementer.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgsvj0(JOBV, M, N, A, LDA, D, SVA, MV, V, LDV, EPS, SFMIN, TOL,
                  NSWEEP, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgsvj0_(const char *jobv, const armpl_int_t *m, const armpl_int_t *n,
             double *a, const armpl_int_t *lda, double *d, double *sva,
             const armpl_int_t *mv, double *v, const armpl_int_t *ldv,
             const double *eps, const double *sfmin, const double *tol,
             const armpl_int_t *nsweep, double *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

JOBV Input parameter.

JOBV is CHARACTER*1

Specifies whether the output from this procedure is used to compute the matrix V: = 'V': the product of the Jacobi rotations is accumulated by postmulyiplying the N-by-N array V. (See the description of V.) = 'A': the product of the Jacobi rotations is accumulated by postmulyiplying the MV-by-N array V. (See the descriptions of MV and V.) = 'N': the Jacobi rotations are not accumulated.

M Input parameter.

M is INTEGER

The number of rows of the input matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the input matrix A. M >= N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, M-by-N matrix A, such that A*diag(D) represents the input matrix. On exit, A_onexit * D_onexit represents the input matrix A*diag(D) post-multiplied by a sequence of Jacobi rotations, where the rotation threshold and the total number of sweeps are given in TOL and NSWEEP, respectively. (See the descriptions of D, TOL and NSWEEP.)

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The array D accumulates the scaling factors from the fast scaled Jacobi rotations. On entry, A*diag(D) represents the input matrix. On exit, A_onexit*diag(D_onexit) represents the input matrix post-multiplied by a sequence of Jacobi rotations, where the rotation threshold and the total number of sweeps are given in TOL and NSWEEP, respectively. (See the descriptions of A, TOL and NSWEEP.)

SVA Input and output parameter.

SVA is DOUBLE PRECISION

SVA is an array, dimension (N). On entry, SVA contains the Euclidean norms of the columns of the matrix A*diag(D). On exit, SVA contains the Euclidean norms of the columns of the matrix onexit*diag(D_onexit).

MV Input parameter.

MV is INTEGER

If JOBV .EQ. 'A', then MV rows of V are post-multipled by a sequence of Jacobi rotations. If JOBV = 'N', then MV is not referenced.

V Input and output parameter.

V is DOUBLE PRECISION

V is an array, dimension (LDV, N). If JOBV .EQ. 'V' then N rows of V are post-multipled by a sequence of Jacobi rotations. If JOBV .EQ. 'A' then MV rows of V are post-multipled by a sequence of Jacobi rotations. If JOBV = 'N', then V is not referenced.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V, LDV >= 1. If JOBV = 'V', LDV .GE. N. If JOBV = 'A', LDV .GE. MV.

EPS Input parameter.

EPS is DOUBLE PRECISION

EPS = DLAMCH('Epsilon')

SFMIN Input parameter.

SFMIN is DOUBLE PRECISION

SFMIN = DLAMCH('Safe Minimum')

TOL Input parameter.

TOL is DOUBLE PRECISION

TOL is the threshold for Jacobi rotations. For a pair A(:,p), A(:,q) of pivot columns, the Jacobi rotation is applied only if DABS(COS(angle(A(:,p),A(:,q)))) .GT. TOL.

NSWEEP Input parameter.

NSWEEP is INTEGER

NSWEEP is the number of sweeps of Jacobi rotations to be performed.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

LWORK is the dimension of WORK. LWORK .GE. M.

INFO Output parameter.

INFO is INTEGER

= 0 : successful exit. < 0 : if INFO = -i, then the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgsvj0*, *sgsvj0* and *zgsvj0*.

4.17.203 dgsvj1

dgsvj1 is called from DGESVJ as a pre-processor and that is its main purpose. It applies Jacobi rotations in the same way as DGESVJ does, but it targets only particular pivots and it does not check convergence (stopping criterion). Few tunning parameters (marked by [TP]) are available for the implementer.

Further Details

dgsvj1 applies few sweeps of Jacobi rotations in the column space of the input M-by-N matrix A. The pivot pairs are taken from the (1,2) off-diagonal block in the corresponding N-by-N Gram matrix A^T * A. The block-entries (tiles) of the (1,2) off-diagonal block are marked by the [x]'s in the following scheme:

```
| *  *  * [x] [x] [x]|
| *  *  * [x] [x] [x]|    Row-cycling in the nblr-by-nblc [x] blocks.
| *  *  * [x] [x] [x]|    Row-cyclic pivoting inside each [x] block.
|[x] [x] [x] *  *  * |
|[x] [x] [x] *  *  * |
|[x] [x] [x] *  *  * |
```

In terms of the columns of A, the first N1 columns are rotated 'against' the remaining N-N1 columns, trying to increase the angle between the corresponding subspaces. The off-diagonal block is N1-by(N-N1) and it is tiled using quadratic tiles of side KBL. Here, KBL is a tunning parmeter. The number of sweeps is given in NSWEEP and the orthogonality threshold is given in TOL.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgsvj1(JOBV, M, N, N1, A, LDA, D, SVA, MV, V, LDV, EPS, SFMIN, TOL,
                  NSWEEP, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dgsvj1_(const char *jobv, const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *n1, double *a, const armpl_int_t *lda,
             double *d, double *sva, const armpl_int_t *mv, double *v,
             const armpl_int_t *ldv, const double *eps, const double *sfmin,
             const double *tol, const armpl_int_t *nsweep, double *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

JOBV Input parameter.

JOBV is CHARACTER*1

Specifies whether the output from this procedure is used to compute the matrix V: = 'V': the product of the Jacobi rotations is accumulated by postmulyiplying the N-by-N array V. (See the description of V.) = 'A': the product of the Jacobi rotations is accumulated by postmulyiplying the MV-by-N array V. (See the descriptions of MV and V.) = 'N': the Jacobi rotations are not accumulated.

M Input parameter.

M is INTEGER

The number of rows of the input matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the input matrix A. M >= N >= 0.

N1 Input parameter.

N1 is INTEGER

N1 specifies the 2 x 2 block partition, the first N1 columns are rotated 'against' the remaining N-N1 columns of A.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, M-by-N matrix A, such that A*diag(D) represents the input matrix. On exit, A_onexit * D_onexit represents the input matrix A*diag(D) post-multiplied by a sequence of Jacobi rotations, where the rotation threshold and the total number of sweeps are given in TOL and NSWEEP, respectively. (See the descriptions of N1, D, TOL and NSWEEP.)

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The array D accumulates the scaling factors from the fast scaled Jacobi rotations. On entry, A*diag(D) represents the input matrix. On exit, A_onexit*diag(D_onexit) represents the input matrix post-multiplied by a sequence of Jacobi rotations, where the rotation threshold and the total number of sweeps are given in TOL and NSWEEP, respectively. (See the descriptions of N1, A, TOL and NSWEEP.)

SVA Input and output parameter.

SVA is DOUBLE PRECISION

SVA is an array, dimension (N). On entry, SVA contains the Euclidean norms of the columns of the matrix A*diag(D). On exit, SVA contains the Euclidean norms of the columns of the matrix onexit*diag(D_onexit).

MV Input parameter.

MV is INTEGER

If JOBV .EQ. 'A', then MV rows of V are post-multipled by a sequence of Jacobi rotations. If JOBV = 'N', then MV is not referenced.

V Input and output parameter.

V is DOUBLE PRECISION

V is an array, dimension (LDV, N). If JOBV .EQ. 'V' then N rows of V are post-multipled by a sequence of Jacobi rotations. If JOBV .EQ. 'A' then MV rows of V are post-multipled by a sequence of Jacobi rotations. If JOBV = 'N', then V is not referenced.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V, LDV >= 1. If JOBV = 'V', LDV .GE. N. If JOBV = 'A', LDV .GE. MV.

EPS Input parameter.

EPS is DOUBLE PRECISION

EPS = DLAMCH('Epsilon')

SFMIN Input parameter.

SFMIN is DOUBLE PRECISION

SFMIN = DLAMCH('Safe Minimum')

TOL Input parameter.

TOL is DOUBLE PRECISION

TOL is the threshold for Jacobi rotations. For a pair A(:,p), A(:,q) of pivot columns, the Jacobi rotation is applied only if DABS(COS(angle(A(:,p),A(:,q)))) .GT. TOL.

NSWEEP Input parameter.

NSWEEP is INTEGER

NSWEEP is the number of sweeps of Jacobi rotations to be performed.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

LWORK is the dimension of WORK. LWORK .GE. M.

INFO Output parameter.

INFO is INTEGER

= 0 : successful exit. < 0 : if INFO = -i, then the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgsvj1*, *sgsvj1* and *zgsvj1*.

4.17.204 dgtts2

dgtts2 solves one of the systems of equations

```
A*X = B  or  A**T*X = B,
```

with a tridiagonal matrix A using the LU factorization computed by DGTTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine dgtts2(ITRANS, N, NRHS, DL, D, DU, DU2, IPIV, B, LDB)
```

C specification:

```
#include "armpl.h"

void dgtts2_(const armpl_int_t *itrans, const armpl_int_t *n,
             const armpl_int_t *nrhs, const double *dl, const double *d,
             const double *du, const double *du2, const armpl_int_t *ipiv,
             double *b, const armpl_int_t *ldb);
```

Parameters

ITRANS Input parameter.

ITRANS is INTEGER

Specifies the form of the system of equations. = 0: A * X = B (No transpose) = 1: A^T * X = B (Transpose) = 2: A^T * X = B (Conjugate transpose = Transpose)

N Input parameter.

N is INTEGER

The order of the matrix A.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

DL Input parameter.

DL is DOUBLE PRECISION

DL is an array, dimension (N-1). The (n-1) multipliers that define the matrix L from the LU factorization of A.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The n diagonal elements of the upper triangular matrix U from the LU factorization of A.

DU Input parameter.

DU is DOUBLE PRECISION

DU is an array, dimension (N-1). The (n-1) elements of the first super-diagonal of U.

DU2 Input parameter.

DU2 is DOUBLE PRECISION

DU2 is an array, dimension (N-2). The (n-2) elements of the second super-diagonal of U.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices; for 1 <= i <= n, row i of the matrix was interchanged with row IPIV(i). IPIV(i) will always be either i or i+1; IPIV(i) = i indicates a row interchange was not required.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the matrix of right hand side vectors B. On exit, B is overwritten by the solution vectors X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Related Information

For this routine in other precisions, please see *cgtts2*, *sgtts2* and *zgtts2*.

4.17.205 disnan

disnan returns .TRUE. if its argument is NaN, and .FALSE. otherwise. To be replaced by the Fortran 2003 intrinsic in the future.

Syntax

Fortran specification:

```
use armpl_library

logical function disnan(DIN)
```

C specification:

```
#include "armpl.h"

armpl_int_t disnan_(const double *din);
```

Parameters

DIN Input parameter.

DIN is DOUBLE PRECISION

Input to test for NaN.

Related Information

For this routine in other precisions, please see *sisnan*.

4.17.206 dla_gbamv

DLA_GBAMV performs one of the matrix-vector operations

```
     y := alpha*abs(A)*abs(x) + beta*abs(y),
or   y := alpha*abs(A)**T*abs(x) + beta*abs(y),
```

where alpha and beta are scalars, x and y are vectors and A is an m by n matrix.

This function is primarily used in calculating error bounds. To protect against underflow during evaluation, components in the resulting vector are perturbed away from zero by (N+1) times the underflow threshold. To prevent unnecessarily large errors for block-structure embedded in general matrices, "symbolically" zero components are not perturbed. A zero entry is considered "symbolic" if all multiplications involved in computing that entry have at least one zero multiplicand.

Syntax

Fortran specification:

```
use armpl_library

subroutine dla_gbamv(TRANS, M, N, KL, KU, ALPHA, AB, LDAB, X, INCX, BETA, Y,
                     INCY)
```

C specification:

```
#include "armpl.h"

void dla_gbamv_(const armpl_int_t *trans, const armpl_int_t *m,
                const armpl_int_t *n, const armpl_int_t *kl,
                const armpl_int_t *ku, const double *alpha, const double *ab,
                const armpl_int_t *ldab, const double *x,
```

(continues on next page)

(continued from previous page)

```
                const armpl_int_t *incx, const double *beta, double *y,
                const armpl_int_t *incy);
```

Parameters

TRANS Input parameter.

TRANS is INTEGER

On entry, TRANS specifies the operation to be performed as follows:

BLAS_NO_TRANS y := alpha*abs(A)*abs(x) + beta*abs(y) BLAS_TRANS y := alpha*abs(A^T)*abs(x) + beta*abs(y) BLAS_CONJ_TRANS y := alpha*abs(A^T)*abs(x) + beta*abs(y)

Unchanged on exit.

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of the matrix A. M must be at least zero. Unchanged on exit.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix A. N must be at least zero. Unchanged on exit.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

ALPHA Input parameter.

ALPHA is DOUBLE PRECISION

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

AB Input parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, n). Before entry, the leading m by n part of the array AB must contain the matrix of coefficients. Unchanged on exit.

LDAB Input parameter.

LDAB is INTEGER

On entry, LDA specifies the first dimension of AB as declared in the calling (sub) program. LDAB must be at least max(1, m). Unchanged on exit.

X Input parameter.

X is DOUBLE PRECISION

X is an array, dimension. (1 + (n - 1)*abs(INCX)) when TRANS = 'N' or 'n' and at least (1 + (m - 1)*abs(INCX)) otherwise. Before entry, the incremented array X must contain the vector x. Unchanged on exit.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

BETA Input parameter.

BETA is DOUBLE PRECISION

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input. Unchanged on exit.

Y Input and output parameter.

Y is DOUBLE PRECISION

Y is an array, dimension. (1 + (m - 1)*abs(INCY)) when TRANS = 'N' or 'n' and at least (1 + (n - 1)*abs(INCY)) otherwise. Before entry with BETA non-zero, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero. Unchanged on exit.

Level 2 Blas routine.

Related Information

For this routine in other precisions, please see *cla_gbamv*, *sla_gbamv* and *zla_gbamv*.

4.17.207 dla_gbrcond

```
DLA_GBRCOND Estimates the Skeel condition number of  op(A) * op2(C)
   where op2 is determined by CMODE as follows
   CMODE =  1    op2(C) = C
   CMODE =  0    op2(C) = I
   CMODE = -1    op2(C) = inv(C)
   The Skeel condition number  cond(A) = norminf( |inv(A)||A| )
   is computed by computing scaling factors R such that
   diag(R)*A*op2(C) is row equilibrated and computing the standard
   infinity-norm condition number.
```

Syntax

Fortran specification:

```
use armpl_library

double precision function dla_gbrcond(TRANS, N, KL, KU, AB, LDAB, AFB, LDAFB,
                                      IPIV, CMODE, C, INFO, WORK, IWORK)
```

C specification:

```
#include "armpl.h"

double dla_gbrcond_(const char *trans, const armpl_int_t *n,
                    const armpl_int_t *kl, const armpl_int_t *ku,
                    const double *ab, const armpl_int_t *ldab,
                    const double *afb, const armpl_int_t *ldafb,
```

(continues on next page)

(continued from previous page)

```
                    const armpl_int_t *ipiv, const armpl_int_t *cmode,
                    const double *c, armpl_int_t *info, double *work,
                    armpl_int_t *iwork, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate Transpose = Transpose)

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

AB Input parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(KU+1+i-j,j) = A(i,j) for max(1,j-KU)<=i<=min(N,j+kl)

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KL+KU+1.

AFB Input parameter.

AFB is DOUBLE PRECISION

AFB is an array, dimension (LDAFB, N). Details of the LU factorization of the band matrix A, as computed by DGBTRF. U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1.

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AFB. LDAFB >= 2*KL+KU+1.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from the factorization A = P*L*U as computed by DGBTRF; row i of the matrix was interchanged with row IPIV(i).

CMODE Input parameter.

CMODE is INTEGER

Determines op2(C) in the formula op(A) * op2(C) as follows: CMODE = 1 op2(C) = C CMODE = 0 op2(C) = I CMODE = -1 op2(C) = inv(C)

C Input parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). The vector C in the formula op(A) * op2(C).

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. i > 0: The ith argument is invalid.

WORK Input parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (5*N).. Workspace.

IWORK Input parameter.

IWORK is INTEGER array, dimension (N).

Workspace.

Related Information

For this routine in other precisions, please see *sla_gbrcond*.

4.17.208 dla_gbrfsx_extended

dla_gbrfsx_extended improves the computed solution to a system of linear equations by performing extra-precise iterative refinement and provides error bounds and backward error estimates for the solution. This subroutine is called by DGBRFSX to perform iterative refinement. In addition to normwise error bound, the code provides maximum componentwise error bound if possible. See comments for ERR_BNDS_NORM and ERR_BNDS_COMP for details of the error bounds. Note that this subroutine is only resonsible for setting the second fields of ERR_BNDS_NORM and ERR_BNDS_COMP.

Syntax

Fortran specification:

```
use armpl_library

subroutine dla_gbrfsx_extended(PREC_TYPE, TRANS_TYPE, N, KL, KU, NRHS, AB,
                               LDAB, AFB, LDAFB, IPIV, COLEQU, C, B, LDB, Y,
                               LDY, BERR_OUT, N_NORMS, ERR_BNDS_NORM,
                               ERR_BNDS_COMP, RES, AYB, DY, Y_TAIL, RCOND,
                               ITHRESH, RTHRESH, DZ_UB, IGNORE_CWISE, INFO)
```

C specification:

```
#include "armpl.h"

void dla_gbrfsx_extended_(const armpl_int_t *prec_type,
                          const armpl_int_t *trans_type, const armpl_int_t *n,
                          const armpl_int_t *kl, const armpl_int_t *ku,
                          const armpl_int_t *nrhs, const double *ab,
                          const armpl_int_t *ldab, const double *afb,
                          const armpl_int_t *ldafb, const armpl_int_t *ipiv,
                          const armpl_int_t *colequ, const double *c,
```

(continues on next page)

(continued from previous page)

```
                                const double *b, const armpl_int_t *ldb, double *y,
                                const armpl_int_t *ldy, double *berr_out,
                                const armpl_int_t *n_norms, double *err_bnds_norm,
                                double *err_bnds_comp, double *res, double *ayb,
                                double *dy, double *y_tail, const double *rcond,
                                const armpl_int_t *ithresh, const double *rthresh,
                                const double *dz_ub,
                                const armpl_int_t *ignore_cwise,
                                armpl_int_t *info);
```

Parameters

PREC_TYPE Input parameter.

PREC_TYPE is INTEGER

Specifies the intermediate precision to be used in refinement. The value is defined by ILAPREC(P) where P is a CHARACTER and P = 'S': Single = 'D': Double = 'I': Indigenous = 'X', 'E': Extra

TRANS_TYPE Input parameter.

TRANS_TYPE is INTEGER

Specifies the transposition operation on A. The value is defined by ILATRANS(T) where T is a CHARACTER and T = 'N': No transpose = 'T': Transpose = 'C': Conjugate transpose

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0

NRHS Input parameter.

NRHS is INTEGER

The number of right-hand-sides, i.e., the number of columns of the matrix B.

AB Input parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). On entry, the N-by-N matrix AB.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDBA >= max(1, N).

AFB Input parameter.

AFB is DOUBLE PRECISION

AFB is an array, dimension (LDAFB, N). The factors L and U from the factorization A = P*L*U as computed by DGBTRF.

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AF. LDAFB >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from the factorization A = P*L*U as computed by DGBTRF; row i of the matrix was interchanged with row IPIV(i).

COLEQU Input parameter.

COLEQU is LOGICAL

If .TRUE. then column equilibration was done to A before calling this routine. This is needed to compute the solution and error bounds correctly.

C Input parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). The column scale factors for A. If COLEQU = .FALSE., C is not accessed. If C is input, each element of C should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). The right-hand-side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Y Input and output parameter.

Y is DOUBLE PRECISION

Y is an array, dimension (LDY, NRHS). On entry, the solution matrix X, as computed by DGBTRS. On exit, the improved solution matrix Y.

LDY Input parameter.

LDY is INTEGER

The leading dimension of the array Y. LDY >= max(1, N).

BERR_OUT Output parameter.

BERR_OUT is DOUBLE PRECISION

BERR_OUT is an array, dimension (NRHS). On exit, BERR_OUT(j) contains the componentwise relative backward error for right-hand-side j from the formula max(i) (abs(RES(i)) / (abs(op(A_s))*abs(Y) + abs(B_s))(i)) where abs(Z) is the componentwise absolute value of the matrix or vector Z. This is computed by DLA_LIN_BERR.

N_NORMS Input parameter.

N_NORMS is INTEGER

Determines which error bounds to return (see ERR_BNDS_NORM and ERR_BNDS_COMP). If N_NORMS >= 1 return normwise error bounds. If N_NORMS >= 2 return componentwise error bounds.

ERR_BNDS_NORM Input and output parameter.

ERR_BNDS_NORM is DOUBLE PRECISION

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

This subroutine is only responsible for setting the second field above. See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Input and output parameter.

ERR_BNDS_COMP is DOUBLE PRECISION

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:,N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

This subroutine is only responsible for setting the second field above. See Lapack Working Note 165 for further details and extra cautions.

RES Input parameter.

RES is DOUBLE PRECISION

RES is an array, dimension (N). Workspace to hold the intermediate residual.

AYB Input parameter.

AYB is DOUBLE PRECISION

AYB is an array, dimension (N). Workspace. This can be the same workspace passed for Y_TAIL.

DY Input parameter.

DY is DOUBLE PRECISION

DY is an array, dimension (N). Workspace to hold the intermediate solution.

Y_TAIL Input parameter.

Y_TAIL is DOUBLE PRECISION

Y_TAIL is an array, dimension (N). Workspace to hold the trailing bits of the intermediate solution.

RCOND Input parameter.

RCOND is DOUBLE PRECISION

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

ITHRESH Input parameter.

ITHRESH is INTEGER

The maximum number of residual computations allowed for refinement. The default is 10. For 'aggressive' set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in ERR_BNDS_NORM and ERR_BNDS_COMP may no longer be trustworthy.

RTHRESH Input parameter.

RTHRESH is DOUBLE PRECISION

Determines when to stop refinement if the error estimate stops decreasing. Refinement will stop when the next solution no longer satisfies norm(dx_{i+1}) < RTHRESH * norm(dx_i) where norm(Z) is the infinity norm of Z. RTHRESH satisfies 0 < RTHRESH <= 1. The default value is 0.5. For 'aggressive' set to 0.9 to permit convergence on extremely ill-conditioned matrices. See LAWN 165 for more details.

DZ_UB Input parameter.

DZ_UB is DOUBLE PRECISION

Determines when to start considering componentwise convergence. Componentwise convergence is only considered after each component of the solution Y is stable, which we definte as the relative change in each component being less than DZ_UB. The default value is 0.25, requiring the first bit to be stable. See LAWN 165 for more details.

IGNORE_CWISE Input parameter.

IGNORE_CWISE is LOGICAL

If .TRUE. then ignore componentwise convergence. Default value is .FALSE..

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. < 0: if INFO = -i, the ith argument to DGBTRS had an illegal value

Related Information

For this routine in other precisions, please see *cla_gbrfsx_extended*, *sla_gbrfsx_extended* and *zla_gbrfsx_extended*.

4.17.209 dla_gbrpvgrw

DLA_GBRPVGRW computes the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If this is much less than 1, the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, estimated condition numbers, and error bounds could be unreliable.

Syntax

Fortran specification:

```
use armpl_library

double precision function dla_gbrpvgrw(N, KL, KU, NCOLS, AB, LDAB, AFB,
                                       LDAFB)
```

C specification:

```
#include "armpl.h"

double dla_gbrpvgrw_(const armpl_int_t *n, const armpl_int_t *kl,
                     const armpl_int_t *ku, const armpl_int_t *ncols,
                     const double *ab, const armpl_int_t *ldab,
                     const double *afb, const armpl_int_t *ldafb);
```

Parameters

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

NCOLS Input parameter.

NCOLS is INTEGER

The number of columns of the matrix A. NCOLS >= 0.

AB Input parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(KU+1+i-j,j) = A(i,j) for max(1,j-KU)<=i<=min(N,j+kl)

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KL+KU+1.

AFB Input parameter.

AFB is DOUBLE PRECISION

AFB is an array, dimension (LDAFB, N). Details of the LU factorization of the band matrix A, as computed by DGBTRF. U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1.

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AFB. LDAFB >= 2*KL+KU+1.

Related Information

For this routine in other precisions, please see *cla_gbrpvgrw*, *sla_gbrpvgrw* and *zla_gbrpvgrw*.

4.17.210 dla_geamv

DLA_GEAMV performs one of the matrix-vector operations

```
     y := alpha*abs(A)*abs(x) + beta*abs(y),
or   y := alpha*abs(A)**T*abs(x) + beta*abs(y),
```

where alpha and beta are scalars, x and y are vectors and A is an m by n matrix.

This function is primarily used in calculating error bounds. To protect against underflow during evaluation, components in the resulting vector are perturbed away from zero by (N+1) times the underflow threshold. To prevent unnecessarily large errors for block-structure embedded in general matrices, "symbolically" zero components are not perturbed. A zero entry is considered "symbolic" if all multiplications involved in computing that entry have at least one zero multiplicand.

Syntax

Fortran specification:

```
use armpl_library

subroutine dla_geamv(TRANS, M, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
```

C specification:

```
#include "armpl.h"

void dla_geamv_(const armpl_int_t *trans, const armpl_int_t *m,
                const armpl_int_t *n, const double *alpha, const double *a,
                const armpl_int_t *lda, const double *x,
                const armpl_int_t *incx, const double *beta, double *y,
                const armpl_int_t *incy);
```

Parameters

TRANS Input parameter.

TRANS is INTEGER

On entry, TRANS specifies the operation to be performed as follows:

BLAS_NO_TRANS y := alpha*abs(A)*abs(x) + beta*abs(y) BLAS_TRANS y := alpha*abs(A^T)*abs(x) + beta*abs(y) BLAS_CONJ_TRANS y := alpha*abs(A^T)*abs(x) + beta*abs(y)

Unchanged on exit.

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of the matrix A. M must be at least zero. Unchanged on exit.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix A. N must be at least zero. Unchanged on exit.

ALPHA Input parameter.

ALPHA is DOUBLE PRECISION

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, n). Before entry, the leading m by n part of the array A must contain the matrix of coefficients. Unchanged on exit.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, m). Unchanged on exit.

X Input parameter.

X is DOUBLE PRECISION

X is an array, dimension. (1 + (n - 1)*abs(INCX)) when TRANS = 'N' or 'n' and at least (1 + (m - 1)*abs(INCX)) otherwise. Before entry, the incremented array X must contain the vector x. Unchanged on exit.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

BETA Input parameter.

BETA is DOUBLE PRECISION

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input. Unchanged on exit.

Y Input and output parameter.

Y is DOUBLE PRECISION

Y is an array,. dimension at least (1 + (m - 1)*abs(INCY)) when TRANS = 'N' or 'n' and at least (1 + (n - 1)*abs(INCY)) otherwise. Before entry with BETA non-zero, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero. Unchanged on exit.

Level 2 Blas routine.

Related Information

For this routine in other precisions, please see *cla_geamv*, *sla_geamv* and *zla_geamv*.

4.17.211 dla_gercond

```
DLA_GERCOND estimates the Skeel condition number of op(A) * op2(C)
   where op2 is determined by CMODE as follows
   CMODE =  1    op2(C) = C
   CMODE =  0    op2(C) = I
   CMODE = -1    op2(C) = inv(C)
   The Skeel condition number cond(A) = norminf( |inv(A)||A| )
   is computed by computing scaling factors R such that
   diag(R)*A*op2(C) is row equilibrated and computing the standard
   infinity-norm condition number.
```

Syntax

Fortran specification:

```
use armpl_library

double precision function dla_gercond(TRANS, N, A, LDA, AF, LDAF, IPIV, CMODE,
                                      C, INFO, WORK, IWORK)
```

C specification:

```
#include "armpl.h"

double dla_gercond_(const char *trans, const armpl_int_t *n, const double *a,
                    const armpl_int_t *lda, const double *af,
                    const armpl_int_t *ldaf, const armpl_int_t *ipiv,
                    const armpl_int_t *cmode, const double *c,
                    armpl_int_t *info, double *work, armpl_int_t *iwork,
                    ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate Transpose = Transpose)

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is DOUBLE PRECISION

AF is an array, dimension (LDAF, N). The factors L and U from the factorization A = P*L*U as computed by DGETRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from the factorization A = P*L*U as computed by DGETRF; row i of the matrix was interchanged with row IPIV(i).

CMODE Input parameter.

CMODE is INTEGER

Determines op2(C) in the formula op(A) * op2(C) as follows: CMODE = 1 op2(C) = C CMODE = 0 op2(C) = I CMODE = -1 op2(C) = inv(C)

C Input parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). The vector C in the formula op(A) * op2(C).

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. i > 0: The ith argument is invalid.

WORK Input parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (3*N).. Workspace.

IWORK Input parameter.

IWORK is INTEGER array, dimension (N).

Workspace.

Related Information

For this routine in other precisions, please see *sla_gercond*.

4.17.212 dla_gerfsx_extended

`dla_gerfsx_extended` improves the computed solution to a system of linear equations by performing extra-precise iterative refinement and provides error bounds and backward error estimates for the solution. This subroutine is called by DGERFSX to perform iterative refinement. In addition to normwise error bound, the code provides maximum componentwise error bound if possible. See comments for ERRS_N and ERRS_C for details of the error bounds. Note that this subroutine is only resonsible for setting the second fields of ERRS_N and ERRS_C.

Syntax

Fortran specification:

```
use armpl_library

subroutine dla_gerfsx_extended(PREC_TYPE, TRANS_TYPE, N, NRHS, A, LDA, AF,
                               LDAF, IPIV, COLEQU, C, B, LDB, Y, LDY,
                               BERR_OUT, N_NORMS, ERRS_N, ERRS_C, RES, AYB,
                               DY, Y_TAIL, RCOND, ITHRESH, RTHRESH, DZ_UB,
                               IGNORE_CWISE, INFO)
```

C specification:

```
#include "armpl.h"

void dla_gerfsx_extended_(const armpl_int_t *prec_type,
                          const armpl_int_t *trans_type, const armpl_int_t *n,
                          const armpl_int_t *nrhs, const double *a,
                          const armpl_int_t *lda, const double *af,
                          const armpl_int_t *ldaf, const armpl_int_t *ipiv,
                          const armpl_int_t *colequ, const double *c,
                          const double *b, const armpl_int_t *ldb, double *y,
                          const armpl_int_t *ldy, double *berr_out,
                          const armpl_int_t *n_norms, double *errs_n,
                          double *errs_c, double *res, double *ayb,
                          double *dy, double *y_tail, const double *rcond,
                          const armpl_int_t *ithresh, const double *rthresh,
                          const double *dz_ub,
                          const armpl_int_t *ignore_cwise,
                          armpl_int_t *info);
```

Parameters

PREC_TYPE Input parameter.

`PREC_TYPE` is `INTEGER`

Specifies the intermediate precision to be used in refinement. The value is defined by ILAPREC(P) where P is a CHARACTER and P = 'S': Single = 'D': Double = 'I': Indigenous = 'X', 'E': Extra

TRANS_TYPE Input parameter.

`TRANS_TYPE` is `INTEGER`

Specifies the transposition operation on `A`. The value is defined by ILATRANS(T) where T is a CHARACTER and T = '`N`': No transpose = '`T`': Transpose = '`C`': Conjugate transpose

N Input parameter.

`N` is `INTEGER`

The number of linear equations, i.e., the order of the matrix `A`. `N` >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right-hand-sides, i.e., the number of columns of the matrix B.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is DOUBLE PRECISION

AF is an array, dimension (LDAF, N). The factors L and U from the factorization A = P*L*U as computed by DGETRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from the factorization A = P*L*U as computed by DGETRF; row i of the matrix was interchanged with row IPIV(i).

COLEQU Input parameter.

COLEQU is LOGICAL

If .TRUE. then column equilibration was done to A before calling this routine. This is needed to compute the solution and error bounds correctly.

C Input parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). The column scale factors for A. If COLEQU = .FALSE., C is not accessed. If C is input, each element of C should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). The right-hand-side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Y Input and output parameter.

Y is DOUBLE PRECISION

Y is an array, dimension (LDY, NRHS). On entry, the solution matrix X, as computed by DGETRS. On exit, the improved solution matrix Y.

LDY Input parameter.

LDY is INTEGER

The leading dimension of the array Y. LDY >= max(1, N).

BERR_OUT Output parameter.

BERR_OUT is DOUBLE PRECISION

BERR_OUT is an array, dimension (NRHS). On exit, BERR_OUT(j) contains the componentwise relative backward error for right-hand-side j from the formula max(i) (abs(RES(i)) / (abs(op(A_s))*abs(Y) + abs(B_s))(i)) where abs(Z) is the componentwise absolute value of the matrix or vector Z. This is computed by DLA_LIN_BERR.

N_NORMS Input parameter.

N_NORMS is INTEGER

Determines which error bounds to return (see ERRS_N and ERRS_C). If N_NORMS >= 1 return normwise error bounds. If N_NORMS >= 2 return componentwise error bounds.

ERRS_N Input and output parameter.

ERRS_N is DOUBLE PRECISION

ERRS_N is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERRS_N(i,:) corresponds to the ith right-hand side.

The second index in ERRS_N(:,err) contains the following three fields: err = 1 “Trust/don’t trust” boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch(‘Epsilon’).

err = 2 “Guaranteed” error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch(‘Epsilon’). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch(‘Epsilon’) to determine if the error estimate is “guaranteed”. These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

This subroutine is only responsible for setting the second field above. See Lapack Working Note 165 for further details and extra cautions.

ERRS_C Input and output parameter.

ERRS_C is DOUBLE PRECISION

ERRS_C is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERRS_C is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:,N_ERR_BNDS) entries are returned.

The first index in ERRS_C(i,:) corresponds to the ith right-hand side.

The second index in ERRS_C(:,err) contains the following three fields: err = 1 “Trust/don’t trust” boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch(‘Epsilon’).

err = 2 “Guaranteed” error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch(‘Epsilon’). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch(‘Epsilon’) to determine if the error estimate is “guaranteed”. These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

This subroutine is only responsible for setting the second field above. See Lapack Working Note 165 for further details and extra cautions.

RES Input parameter.

RES is DOUBLE PRECISION

RES is an array, dimension (N). Workspace to hold the intermediate residual.

AYB Input parameter.

AYB is DOUBLE PRECISION

AYB is an array, dimension (N). Workspace. This can be the same workspace passed for Y_TAIL.

DY Input parameter.

DY is DOUBLE PRECISION

DY is an array, dimension (N). Workspace to hold the intermediate solution.

Y_TAIL Input parameter.

Y_TAIL is DOUBLE PRECISION

Y_TAIL is an array, dimension (N). Workspace to hold the trailing bits of the intermediate solution.

RCOND Input parameter.

RCOND is DOUBLE PRECISION

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

ITHRESH Input parameter.

ITHRESH is INTEGER

The maximum number of residual computations allowed for refinement. The default is 10. For ‘aggressive’ set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in ERRS_N and ERRS_C may no longer be trustworthy.

RTHRESH Input parameter.

RTHRESH is DOUBLE PRECISION

Determines when to stop refinement if the error estimate stops decreasing. Refinement will stop when the next solution no longer satisfies norm(dx_{i+1}) < RTHRESH * norm(dx_i) where norm(Z) is the infinity norm of Z. RTHRESH satisfies 0 < RTHRESH <= 1. The default value is 0.5. For ‘aggressive’ set to 0.9 to permit convergence on extremely ill-conditioned matrices. See LAWN 165 for more details.

DZ_UB Input parameter.

DZ_UB is DOUBLE PRECISION

Determines when to start considering componentwise convergence. Componentwise convergence is only considered after each component of the solution `Y` is stable, which we definte as the relative change in each component being less than `DZ_UB`. The default value is 0.25, requiring the first bit to be stable. See LAWN 165 for more details.

IGNORE_CWISE Input parameter.

`IGNORE_CWISE` is `LOGICAL`

If .TRUE. then ignore componentwise convergence. Default value is .FALSE..

INFO Output parameter.

`INFO` is `INTEGER`

= 0: Successful exit. < 0: if `INFO` = -i, the ith argument to DGETRS had an illegal value

Related Information

For this routine in other precisions, please see *cla_gerfsx_extended*, *sla_gerfsx_extended* and *zla_gerfsx_extended*.

4.17.213 dla_gerpvgrw

DLA_GERPVGRW computes the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If this is much less than 1, the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, estimated condition numbers, and error bounds could be unreliable.

Syntax

Fortran specification:

```
use armpl_library

double precision function dla_gerpvgrw(N, NCOLS, A, LDA, AF, LDAF)
```

C specification:

```
#include "armpl.h"

double dla_gerpvgrw_(const armpl_int_t *n, const armpl_int_t *ncols,
                     const double *a, const armpl_int_t *lda,
                     const double *af, const armpl_int_t *ldaf);
```

Parameters

N Input parameter.

`N` is `INTEGER`

The number of linear equations, i.e., the order of the matrix `A`. `N` >= 0.

NCOLS Input parameter.

`NCOLS` is `INTEGER`

The number of columns of the matrix `A`. `NCOLS` >= 0.

A Input parameter.

`A` is `DOUBLE PRECISION`

`A` is an array, dimension (`LDA`, `N`). On entry, the N-by-N matrix `A`.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is DOUBLE PRECISION

AF is an array, dimension (LDAF, N). The factors L and U from the factorization A = P*L*U as computed by DGETRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

Related Information

For this routine in other precisions, please see *cla_gerpvgrw*, *sla_gerpvgrw* and *zla_gerpvgrw*.

4.17.214 dla_lin_berr

```
DLA_LIN_BERR computes component-wise relative backward error from
   the formula
       max(i) ( abs(R(i)) / ( abs(op(A_s))*abs(Y) + abs(B_s) )(i) )
   where abs(Z) is the component-wise absolute value of the matrix
   or vector Z.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dla_lin_berr(N, NZ, NRHS, RES, AYB, BERR)
```

C specification:

```
#include "armpl.h"

void dla_lin_berr_(const armpl_int_t *n, const armpl_int_t *nz,
                   const armpl_int_t *nrhs, const double *res,
                   const double *ayb, double *berr);
```

Parameters

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NZ Input parameter.

NZ is INTEGER

We add (NZ+1)*SLAMCH('Safe minimum') to R(i) in the numerator to guard against spuriously zero residuals. Default value is N.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices AYB, RES, and BERR. NRHS >= 0.

RES Input parameter.

RES is DOUBLE PRECISION

RES is an array, dimension (N, NRHS). The residual matrix, i.e., the matrix R in the relative backward error formula above.

AYB Input parameter.

AYB is DOUBLE PRECISION

AYB is an array, dimension (N, NRHS). The denominator in the relative backward error formula above, i.e., the matrix abs(op(A_s))*abs(Y) + abs(B_s). The matrices A, Y, and B are from iterative refinement (see dla_gerfsx_extended.f).

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The component-wise relative backward error from the formula above.

Related Information

For this routine in other precisions, please see *cla_lin_berr*, *sla_lin_berr* and *zla_lin_berr*.

4.17.215 dla_porcond

```
DLA_PORCOND Estimates the Skeel condition number of  op(A) * op2(C)
   where op2 is determined by CMODE as follows
   CMODE =  1    op2(C) = C
   CMODE =  0    op2(C) = I
   CMODE = -1    op2(C) = inv(C)
   The Skeel condition number  cond(A) = norminf( |inv(A)||A| )
   is computed by computing scaling factors R such that
   diag(R)*A*op2(C) is row equilibrated and computing the standard
   infinity-norm condition number.
```

Syntax

Fortran specification:

```
use armpl_library

double precision function dla_porcond(UPLO, N, A, LDA, AF, LDAF, CMODE, C,
                                      INFO, WORK, IWORK)
```

C specification:

```
#include "armpl.h"

double dla_porcond_(const char *uplo, const armpl_int_t *n, const double *a,
                    const armpl_int_t *lda, const double *af,
                    const armpl_int_t *ldaf, const armpl_int_t *cmode,
                    const double *c, armpl_int_t *info, double *work,
                    armpl_int_t *iwork, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is DOUBLE PRECISION

AF is an array, dimension (LDAF, N). The triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , as computed by DPOTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

CMODE Input parameter.

CMODE is INTEGER

Determines op2(C) in the formula op(A) * op2(C) as follows: CMODE = 1 op2(C) = C CMODE = 0 op2(C) = I CMODE = -1 op2(C) = inv(C)

C Input parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). The vector C in the formula op(A) * op2(C).

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. i > 0: The ith argument is invalid.

WORK Input parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (3*N).. Workspace.

IWORK Input parameter.

IWORK is INTEGER array, dimension (N).

Workspace.

Related Information

For this routine in other precisions, please see *sla_porcond*.

4.17.216 dla_porfsx_extended

dla_porfsx_extended improves the computed solution to a system of linear equations by performing extra-precise iterative refinement and provides error bounds and backward error estimates for the solution. This subroutine is called by DPORFSX to perform iterative refinement. In addition to normwise error bound, the code provides maximum componentwise error bound if possible. See comments for ERR_BNDS_NORM and ERR_BNDS_COMP for details of the error bounds. Note that this subroutine is only resonsible for setting the second fields of ERR_BNDS_NORM and ERR_BNDS_COMP.

Syntax

Fortran specification:

```
use armpl_library

subroutine dla_porfsx_extended(PREC_TYPE, UPLO, N, NRHS, A, LDA, AF, LDAF,
                               COLEQU, C, B, LDB, Y, LDY, BERR_OUT, N_NORMS,
                               ERR_BNDS_NORM, ERR_BNDS_COMP, RES, AYB, DY,
                               Y_TAIL, RCOND, ITHRESH, RTHRESH, DZ_UB,
                               IGNORE_CWISE, INFO)
```

C specification:

```
#include "armpl.h"

void dla_porfsx_extended_(const armpl_int_t *prec_type, const char *uplo,
                          const armpl_int_t *n, const armpl_int_t *nrhs,
                          const double *a, const armpl_int_t *lda,
                          const double *af, const armpl_int_t *ldaf,
                          const armpl_int_t *colequ, const double *c,
                          const double *b, const armpl_int_t *ldb, double *y,
                          const armpl_int_t *ldy, double *berr_out,
                          const armpl_int_t *n_norms, double *err_bnds_norm,
                          double *err_bnds_comp, double *res, double *ayb,
                          double *dy, double *y_tail, const double *rcond,
                          const armpl_int_t *ithresh, const double *rthresh,
                          const double *dz_ub,
                          const armpl_int_t *ignore_cwise, armpl_int_t *info,
                          ... );
```

Parameters

PREC_TYPE Input parameter.

PREC_TYPE is INTEGER

Specifies the intermediate precision to be used in refinement. The value is defined by ILAPREC(P) where P is a CHARACTER and P = 'S': Single = 'D': Double = 'I': Indigenous = 'X', 'E': Extra

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right-hand-sides, i.e., the number of columns of the matrix B.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is DOUBLE PRECISION

AF is an array, dimension (LDAF, N). The triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , as computed by DPOTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

COLEQU Input parameter.

COLEQU is LOGICAL

If .TRUE. then column equilibration was done to A before calling this routine. This is needed to compute the solution and error bounds correctly.

C Input parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). The column scale factors for A. If COLEQU = .FALSE., C is not accessed. If C is input, each element of C should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). The right-hand-side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Y Input and output parameter.

Y is DOUBLE PRECISION

Y is an array, dimension (LDY, NRHS). On entry, the solution matrix X, as computed by DPOTRS. On exit, the improved solution matrix Y.

LDY Input parameter.

LDY is INTEGER

The leading dimension of the array Y. LDY >= max(1, N).

BERR_OUT Output parameter.

BERR_OUT is DOUBLE PRECISION

BERR_OUT is an array, dimension (NRHS). On exit, BERR_OUT(j) contains the componentwise relative backward error for right-hand-side j from the formula max(i) (abs(RES(i)) / (abs(op(A_s))*abs(Y) + abs(B_s)

)(i)) where abs(Z) is the componentwise absolute value of the matrix or vector Z. This is computed by DLA_LIN_BERR.

N_NORMS Input parameter.

N_NORMS is INTEGER

Determines which error bounds to return (see ERR_BNDS_NORM and ERR_BNDS_COMP). If N_NORMS >= 1 return normwise error bounds. If N_NORMS >= 2 return componentwise error bounds.

ERR_BNDS_NORM Input and output parameter.

ERR_BNDS_NORM is DOUBLE PRECISION

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

This subroutine is only responsible for setting the second field above. See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Input and output parameter.

ERR_BNDS_COMP is DOUBLE PRECISION

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:,N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

This subroutine is only responsible for setting the second field above. See Lapack Working Note 165 for further details and extra cautions.

RES Input parameter.

RES is DOUBLE PRECISION

RES is an array, dimension (N). Workspace to hold the intermediate residual.

AYB Input parameter.

AYB is DOUBLE PRECISION

AYB is an array, dimension (N). Workspace. This can be the same workspace passed for Y_TAIL.

DY Input parameter.

DY is DOUBLE PRECISION

DY is an array, dimension (N). Workspace to hold the intermediate solution.

Y_TAIL Input parameter.

Y_TAIL is DOUBLE PRECISION

Y_TAIL is an array, dimension (N). Workspace to hold the trailing bits of the intermediate solution.

RCOND Input parameter.

RCOND is DOUBLE PRECISION

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

ITHRESH Input parameter.

ITHRESH is INTEGER

The maximum number of residual computations allowed for refinement. The default is 10. For 'aggressive' set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in ERR_BNDS_NORM and ERR_BNDS_COMP may no longer be trustworthy.

RTHRESH Input parameter.

RTHRESH is DOUBLE PRECISION

Determines when to stop refinement if the error estimate stops decreasing. Refinement will stop when the next solution no longer satisfies norm(dx_{i+1}) < RTHRESH * norm(dx_i) where norm(Z) is the infinity norm of Z. RTHRESH satisfies 0 < RTHRESH <= 1. The default value is 0.5. For 'aggressive' set to 0.9 to permit convergence on extremely ill-conditioned matrices. See LAWN 165 for more details.

DZ_UB Input parameter.

DZ_UB is DOUBLE PRECISION

Determines when to start considering componentwise convergence. Componentwise convergence is only considered after each component of the solution Y is stable, which we definte as the relative change in each component being less than DZ_UB. The default value is 0.25, requiring the first bit to be stable. See LAWN 165 for more details.

IGNORE_CWISE Input parameter.

IGNORE_CWISE is LOGICAL

If .TRUE. then ignore componentwise convergence. Default value is .FALSE..

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. < 0: if INFO = -i, the ith argument to DPOTRS had an illegal value

Related Information

For this routine in other precisions, please see *cla_porfsx_extended*, *sla_porfsx_extended* and *zla_porfsx_extended*.

4.17.217 dla_porpvgrw

DLA_PORPVGRW computes the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If this is much less than 1, the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, estimated condition numbers, and error bounds could be unreliable.

Syntax

Fortran specification:

```
use armpl_library

double precision function dla_porpvgrw(UPLO, NCOLS, A, LDA, AF, LDAF, WORK)
```

C specification:

```
#include "armpl.h"

double dla_porpvgrw_(const char *uplo, const armpl_int_t *ncols,
                     const double *a, const armpl_int_t *lda,
                     const double *af, const armpl_int_t *ldaf, double *work,
                     ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

NCOLS Input parameter.

NCOLS is INTEGER

The number of columns of the matrix A. NCOLS >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA,N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1,N).

AF Input parameter.

AF is DOUBLE PRECISION

AF is an array, dimension (LDAF,N). The triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , as computed by DPOTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1,N).

WORK Input parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (2*N) .

Related Information

For this routine in other precisions, please see *cla_porpvgrw*, *sla_porpvgrw* and *zla_porpvgrw*.

4.17.218 dla_syamv

DLA_SYAMV performs the matrix-vector operation

```
y := alpha*abs(A)*abs(x) + beta*abs(y),
```

where alpha and beta are scalars, x and y are vectors and A is an n by n symmetric matrix.

This function is primarily used in calculating error bounds. To protect against underflow during evaluation, components in the resulting vector are perturbed away from zero by (N+1) times the underflow threshold. To prevent unnecessarily large errors for block-structure embedded in general matrices, "symbolically" zero components are not perturbed. A zero entry is considered "symbolic" if all multiplications involved in computing that entry have at least one zero multiplicand.

Syntax

Fortran specification:

```
use armpl_library

subroutine dla_syamv(UPLO, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
```

C specification:

```
#include "armpl.h"

void dla_syamv_(const armpl_int_t *uplo, const armpl_int_t *n,
                const double *alpha, const double *a, const armpl_int_t *lda,
                const double *x, const armpl_int_t *incx, const double *beta,
                double *y, const armpl_int_t *incy);
```

Parameters

UPLO Input parameter.

UPLO is INTEGER

On entry, UPLO specifies whether the upper or lower triangular part of the array A is to be referenced as follows:

UPLO = BLAS_UPPER Only the upper triangular part of A is to be referenced.

UPLO = BLAS_LOWER Only the lower triangular part of A is to be referenced.

Unchanged on exit.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix A. N must be at least zero. Unchanged on exit.

ALPHA Input parameter.

ALPHA is DOUBLE PRECISION .

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, n).. Before entry, the leading m by n part of the array A must contain the matrix of coefficients. Unchanged on exit.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, n). Unchanged on exit.

X Input parameter.

X is DOUBLE PRECISION

X is an array, dimension. (1 + (n - 1)*abs(INCX)) Before entry, the incremented array X must contain the vector x. Unchanged on exit.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

BETA Input parameter.

BETA is DOUBLE PRECISION .

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input. Unchanged on exit.

Y Input and output parameter.

Y is DOUBLE PRECISION

Y is an array, dimension. (1 + (n - 1)*abs(INCY)) Before entry with BETA non-zero, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero. Unchanged on exit.

Related Information

For this routine in other precisions, please see *cla_syamv*, *sla_syamv* and *zla_syamv*.

4.17.219 dla_syrcond

```
DLA_SYRCOND estimates the Skeel condition number of  op(A) * op2(C)
   where op2 is determined by CMODE as follows
   CMODE =  1    op2(C) = C
   CMODE =  0    op2(C) = I
   CMODE = -1    op2(C) = inv(C)
   The Skeel condition number cond(A) = norminf( |inv(A)||A| )
   is computed by computing scaling factors R such that
   diag(R)*A*op2(C) is row equilibrated and computing the standard
   infinity-norm condition number.
```

Syntax

Fortran specification:

```
use armpl_library

double precision function dla_syrcond(UPLO, N, A, LDA, AF, LDAF, IPIV, CMODE,
                                      C, INFO, WORK, IWORK)
```

C specification:

```
#include "armpl.h"

double dla_syrcond_(const char *uplo, const armpl_int_t *n, const double *a,
                    const armpl_int_t *lda, const double *af,
                    const armpl_int_t *ldaf, const armpl_int_t *ipiv,
                    const armpl_int_t *cmode, const double *c,
                    armpl_int_t *info, double *work, armpl_int_t *iwork,
                    ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is DOUBLE PRECISION

AF is an array, dimension (LDAF, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by DSYTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by DSYTRF.

CMODE Input parameter.

CMODE is INTEGER

Determines op2(C) in the formula op(A) * op2(C) as follows: CMODE = 1 op2(C) = C CMODE = 0 op2(C) = I CMODE = -1 op2(C) = inv(C)

C Input parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). The vector C in the formula op(A) * op2(C).

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. i > 0: The ith argument is invalid.

WORK Input parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (3*N).. Workspace.

IWORK Input parameter.

IWORK is INTEGER array, dimension (N).

Workspace.

Related Information

For this routine in other precisions, please see *sla_syrcond*.

4.17.220 dla_syrfsx_extended

dla_syrfsx_extended improves the computed solution to a system of linear equations by performing extra-precise iterative refinement and provides error bounds and backward error estimates for the solution. This subroutine is called by DSYRFSX to perform iterative refinement. In addition to normwise error bound, the code provides maximum componentwise error bound if possible. See comments for ERR_BNDS_NORM and ERR_BNDS_COMP for details of the error bounds. Note that this subroutine is only resonsible for setting the second fields of ERR_BNDS_NORM and ERR_BNDS_COMP.

Syntax

Fortran specification:

```
use armpl_library

subroutine dla_syrfsx_extended(PREC_TYPE, UPLO, N, NRHS, A, LDA, AF, LDAF,
                               IPIV, COLEQU, C, B, LDB, Y, LDY, BERR_OUT,
                               N_NORMS, ERR_BNDS_NORM, ERR_BNDS_COMP, RES,
```

(continues on next page)

(continued from previous page)

```
                                AYB, DY, Y_TAIL, RCOND, ITHRESH, RTHRESH,
                                DZ_UB, IGNORE_CWISE, INFO)
```

C specification:

```
#include "armpl.h"

void dla_syrfsx_extended_(const armpl_int_t *prec_type, const char *uplo,
                          const armpl_int_t *n, const armpl_int_t *nrhs,
                          const double *a, const armpl_int_t *lda,
                          const double *af, const armpl_int_t *ldaf,
                          const armpl_int_t *ipiv, const armpl_int_t *colequ,
                          const double *c, const double *b,
                          const armpl_int_t *ldb, double *y,
                          const armpl_int_t *ldy, double *berr_out,
                          const armpl_int_t *n_norms, double *err_bnds_norm,
                          double *err_bnds_comp, double *res, double *ayb,
                          double *dy, double *y_tail, const double *rcond,
                          const armpl_int_t *ithresh, const double *rthresh,
                          const double *dz_ub,
                          const armpl_int_t *ignore_cwise, armpl_int_t *info,
                          ... );
```

Parameters

PREC_TYPE Input parameter.

PREC_TYPE is INTEGER

Specifies the intermediate precision to be used in refinement. The value is defined by ILAPREC(P) where P is a CHARACTER and P = 'S': Single = 'D': Double = 'I': Indigenous = 'X', 'E': Extra

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right-hand-sides, i.e., the number of columns of the matrix B.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is DOUBLE PRECISION

AF is an array, dimension (LDAF, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by DSYTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by DSYTRF.

COLEQU Input parameter.

COLEQU is LOGICAL

If .TRUE. then column equilibration was done to A before calling this routine. This is needed to compute the solution and error bounds correctly.

C Input parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). The column scale factors for A. If COLEQU = .FALSE., C is not accessed. If C is input, each element of C should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). The right-hand-side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Y Input and output parameter.

Y is DOUBLE PRECISION

Y is an array, dimension (LDY, NRHS). On entry, the solution matrix X, as computed by DSYTRS. On exit, the improved solution matrix Y.

LDY Input parameter.

LDY is INTEGER

The leading dimension of the array Y. LDY >= max(1, N).

BERR_OUT Output parameter.

BERR_OUT is DOUBLE PRECISION

BERR_OUT is an array, dimension (NRHS). On exit, BERR_OUT(j) contains the componentwise relative backward error for right-hand-side j from the formula max(i) (abs(RES(i)) / (abs(op(A_s))*abs(Y) + abs(B_s))(i)) where abs(Z) is the componentwise absolute value of the matrix or vector Z. This is computed by DLA_LIN_BERR.

N_NORMS Input parameter.

N_NORMS is INTEGER

Determines which error bounds to return (see ERR_BNDS_NORM and ERR_BNDS_COMP). If N_NORMS >= 1 return normwise error bounds. If N_NORMS >= 2 return componentwise error bounds.

ERR_BNDS_NORM Input and output parameter.

ERR_BNDS_NORM is DOUBLE PRECISION

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

This subroutine is only responsible for setting the second field above. See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Input and output parameter.

ERR_BNDS_COMP is DOUBLE PRECISION

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:,N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

This subroutine is only responsible for setting the second field above. See Lapack Working Note 165 for further details and extra cautions.

RES Input parameter.

RES is DOUBLE PRECISION

RES is an array, dimension (N). Workspace to hold the intermediate residual.

AYB Input parameter.

AYB is DOUBLE PRECISION

AYB is an array, dimension (N). Workspace. This can be the same workspace passed for Y_TAIL.

DY Input parameter.

DY is DOUBLE PRECISION

DY is an array, dimension (N). Workspace to hold the intermediate solution.

Y_TAIL Input parameter.

Y_TAIL is DOUBLE PRECISION

Y_TAIL is an array, dimension (N). Workspace to hold the trailing bits of the intermediate solution.

RCOND Input parameter.

RCOND is DOUBLE PRECISION

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

ITHRESH Input parameter.

ITHRESH is INTEGER

The maximum number of residual computations allowed for refinement. The default is 10. For 'aggressive' set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in ERR_BNDS_NORM and ERR_BNDS_COMP may no longer be trustworthy.

RTHRESH Input parameter.

RTHRESH is DOUBLE PRECISION

Determines when to stop refinement if the error estimate stops decreasing. Refinement will stop when the next solution no longer satisfies norm(dx_{i+1}) < RTHRESH * norm(dx_i) where norm(Z) is the infinity norm of Z. RTHRESH satisfies 0 < RTHRESH <= 1. The default value is 0.5. For 'aggressive' set to 0.9 to permit convergence on extremely ill-conditioned matrices. See LAWN 165 for more details.

DZ_UB Input parameter.

DZ_UB is DOUBLE PRECISION

Determines when to start considering componentwise convergence. Componentwise convergence is only considered after each component of the solution Y is stable, which we definte as the relative change in each component being less than DZ_UB. The default value is 0.25, requiring the first bit to be stable. See LAWN 165 for more details.

IGNORE_CWISE Input parameter.

IGNORE_CWISE is LOGICAL

If .TRUE. then ignore componentwise convergence. Default value is .FALSE..

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. < 0: if INFO = -i, the ith argument to DLA_SYRFSX_EXTENDED had an illegal value

Related Information

For this routine in other precisions, please see *cla_syrfsx_extended*, *sla_syrfsx_extended* and *zla_syrfsx_extended*.

4.17.221 dla_syrpvgrw

DLA_SYRPVGRW computes the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If this is much less than 1, the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, estimated condition numbers, and error bounds could be unreliable.

Syntax

Fortran specification:

```
use armpl_library

double precision function dla_syrpvgrw(UPLO, N, INFO, A, LDA, AF, LDAF, IPIV,
                                       WORK)
```

C specification:

```
#include "armpl.h"

double dla_syrpvgrw_(const char *uplo, const armpl_int_t *n,
                     const armpl_int_t *info, const double *a,
                     const armpl_int_t *lda, const double *af,
                     const armpl_int_t *ldaf, const armpl_int_t *ipiv,
                     double *work, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

INFO Input parameter.

INFO is INTEGER

The value of INFO returned from DSYTRF, .i.e., the pivot in column INFO is exactly 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is DOUBLE PRECISION

AF is an array, dimension (LDAF, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by DSYTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by DSYTRF.

WORK Input parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (2*N) .

Related Information

For this routine in other precisions, please see *cla_syrpvgrw*, *sla_syrpvgrw* and *zla_syrpvgrw*.

4.17.222 dla_wwaddw

```
DLA_WWADDW adds a vector W into a doubled-single vector (X, Y).

   This works for all extant IBM's hex and binary floating point
   arithmetics, but not for decimal.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dla_wwaddw(N, X, Y, W)
```

C specification:

```
#include "armpl.h"

void dla_wwaddw_(const armpl_int_t *n, double *x, double *y,
                 const double *w);
```

Parameters

N Input parameter.

N is INTEGER

The length of vectors X, Y, and W.

X Input and output parameter.

X is DOUBLE PRECISION

X is an array, dimension (N). The first part of the doubled-single accumulation vector.

Y Input and output parameter.

Y is DOUBLE PRECISION

Y is an array, dimension (N). The second part of the doubled-single accumulation vector.

W Input parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). The vector to be added.

Related Information

For this routine in other precisions, please see *cla_wwaddw*, *sla_wwaddw* and *zla_wwaddw*.

4.17.223 dlabrd

dlabrd reduces the first NB rows and columns of a real general m by n matrix A to upper or lower bidiagonal form by an orthogonal transformation Q^T * A * P, and returns the matrices X and Y which are needed to apply the transformation to the unreduced part of A.

If m >= n, A is reduced to upper bidiagonal form; if m < n, to lower bidiagonal form.

This is an auxiliary routine called by DGEBRD

Syntax

Fortran specification:

```
use armpl_library

subroutine dlabrd(M, N, NB, A, LDA, D, E, TAUQ, TAUP, X, LDX, Y, LDY)
```

C specification:

```
#include "armpl.h"

void dlabrd_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *nb, double *a, const armpl_int_t *lda,
             double *d, double *e, double *tauq, double *taup, double *x,
             const armpl_int_t *ldx, double *y, const armpl_int_t *ldy);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows in the matrix A.

N Input parameter.

N is INTEGER

The number of columns in the matrix A.

NB Input parameter.

NB is INTEGER

The number of leading rows and columns of A to be reduced.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the m by n general matrix to be reduced. On exit, the first NB rows and columns of the matrix are overwritten; the rest of the array is unchanged. If m >= n, elements on and below the diagonal in the first NB columns, with the array TAUQ, represent the orthogonal matrix Q as a product of elementary reflectors; and elements above the diagonal in the first NB rows, with the array TAUP, represent the orthogonal matrix P as a product of elementary reflectors. If m < n, elements below the diagonal in the first NB columns, with the array TAUQ, represent the orthogonal matrix Q as a product of elementary reflectors, and elements on and above the diagonal in the first NB rows, with the array TAUP, represent the orthogonal matrix P as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

D Output parameter.

D is DOUBLE PRECISION

D is an array, dimension (NB). The diagonal elements of the first NB rows and columns of the reduced matrix. D(i) = A(i,i).

E Output parameter.

E is DOUBLE PRECISION

E is an array, dimension (NB). The off-diagonal elements of the first NB rows and columns of the reduced matrix.

TAUQ Output parameter.

TAUQ is DOUBLE PRECISION

TAUQ is an array, dimension (NB). The scalar factors of the elementary reflectors which represent the orthogonal matrix Q. See Further Details.

TAUP Output parameter.

TAUP is DOUBLE PRECISION

TAUP is an array, dimension (NB). The scalar factors of the elementary reflectors which represent the orthogonal matrix P. See Further Details.

X Output parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, NB). The m-by-nb matrix X required to update the unreduced part of A.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, M).

Y Output parameter.

Y is DOUBLE PRECISION

Y is an array, dimension (LDY, NB). The n-by-nb matrix Y required to update the unreduced part of A.

LDY Input parameter.

LDY is INTEGER

The leading dimension of the array Y. LDY >= max(1, N).

Related Information

For this routine in other precisions, please see *clabrd*, *slabrd* and *zlabrd*.

4.17.224 dlacn2

dlacn2 estimates the 1-norm of a square, real matrix A. Reverse communication is used for evaluating matrix-vector products.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlacn2(N, V, X, ISGN, EST, KASE, ISAVE)
```

C specification:

```
#include "armpl.h"

void dlacn2_(const armpl_int_t *n, double *v, double *x, armpl_int_t *isgn,
             double *est, armpl_int_t *kase, armpl_int_t *isave);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix. N >= 1.

V Output parameter.

V is DOUBLE PRECISION

V is an array, dimension (N). On the final return, V = A*W, where EST = norm(V)/norm(W) (W is not returned).

X Input and output parameter.

X is DOUBLE PRECISION

X is an array, dimension (N). On an intermediate return, X should be overwritten by A * X, if KASE=1, A^T * X, if KASE=2, and DLACN2 must be re-called with all the other parameters unchanged.

ISGN Output parameter.

ISGN is INTEGER array, dimension (N)

EST Input and output parameter.

EST is DOUBLE PRECISION

On entry with KASE = 1 or 2 and ISAVE(1) = 3, EST should be unchanged from the previous call to DLACN2. On exit, EST is an estimate (a lower bound) for norm(A).

KASE Input and output parameter.

KASE is INTEGER

On the initial call to DLACN2, KASE should be 0. On an intermediate return, KASE will be 1 or 2, indicating whether X should be overwritten by A * X or A^T * X. On the final return from DLACN2, KASE will again be 0.

ISAVE Input and output parameter.

ISAVE is INTEGER array, dimension (3)

ISAVE is used to save variables between calls to DLACN2

Related Information

For this routine in other precisions, please see *clacn2*, *slacn2* and *zlacn2*. It also exists with a native C interface as *LAPACKE_dlacn2*.

4.17.225 dlacon

dlacon estimates the 1-norm of a square, real matrix A. Reverse communication is used for evaluating matrix-vector products.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlacon(N, V, X, ISGN, EST, KASE)
```

C specification:

```
#include "armpl.h"

void dlacon_(const armpl_int_t *n, double *v, double *x, armpl_int_t *isgn,
             double *est, armpl_int_t *kase);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix. N >= 1.

V Output parameter.

V is DOUBLE PRECISION

V is an array, dimension (N). On the final return, V = A*W, where EST = norm(V)/norm(W) (W is not returned).

X Input and output parameter.

X is DOUBLE PRECISION

X is an array, dimension (N). On an intermediate return, X should be overwritten by A * X, if KASE=1, A^T * X, if KASE=2, and DLACON must be re-called with all the other parameters unchanged.

ISGN Output parameter.

ISGN is INTEGER array, dimension (N)

EST Input and output parameter.

EST is DOUBLE PRECISION

On entry with KASE = 1 or 2 and JUMP = 3, EST should be unchanged from the previous call to DLACON. On exit, EST is an estimate (a lower bound) for norm(A).

KASE Input and output parameter.

KASE is INTEGER

On the initial call to DLACON, KASE should be 0. On an intermediate return, KASE will be 1 or 2, indicating whether X should be overwritten by A * X or A^T * X. On the final return from DLACON, KASE will again be 0.

Related Information

For this routine in other precisions, please see *clacon*, *slacon* and *zlacon*.

4.17.226 dlacpy

dlacpy copies all or part of a two-dimensional matrix A to another matrix B.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlacpy(UPLO, M, N, A, LDA, B, LDB)
```

C specification:

```
#include "armpl.h"

void dlacpy_(const char *uplo, const armpl_int_t *m, const armpl_int_t *n,
             const double *a, const armpl_int_t *lda, double *b,
             const armpl_int_t *ldb, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies the part of the matrix A to be copied to B. = 'U': Upper triangular part = 'L': Lower triangular part Otherwise: All of the matrix A

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). The m by n matrix A. If UPLO = 'U', only the upper triangle or trapezoid is accessed; if UPLO = 'L', only the lower triangle or trapezoid is accessed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). On exit, B = A in the locations specified by UPLO.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M).

Related Information

For this routine in other precisions, please see *clacpy*, *slacpy* and *zlacpy*. It also exists with a native C interface as *LAPACKE_dlacpy*.

4.17.227 dladiv

dladiv performs complex division in real arithmetic

```
            a + i*b
p + i*q = ---------
            c + i*d
```

The algorithm is due to Michael Baudin and Robert L. Smith and can be found in the paper "A Robust Complex Division in Scilab"

Syntax

Fortran specification:

```
use armpl_library

subroutine dladiv(A, B, C, D, P, Q)
```

C specification:

```
#include "armpl.h"

void dladiv_(const double *a, const double *b, const double *c,
             const double *d, double *p, double *q);
```

Parameters

A Input parameter.

A is DOUBLE PRECISION

B Input parameter.

B is DOUBLE PRECISION

C Input parameter.

C is DOUBLE PRECISION

D Input parameter.

D is DOUBLE PRECISION

The scalars a, b, c, and d in the above expression.

P Output parameter.

P is DOUBLE PRECISION

Q Output parameter.

Q is DOUBLE PRECISION

The scalars p and q in the above expression.

Related Information

For this routine in other precisions, please see *cladiv*, *sladiv* and *zladiv*.

4.17.228 dlae2

dlae2 computes the eigenvalues of a 2-by-2 symmetric matrix

```
[  A   B  ]
[  B   C  ].
```

On return, RT1 is the eigenvalue of larger absolute value, and RT2 is the eigenvalue of smaller absolute value.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlae2(A, B, C, RT1, RT2)
```

C specification:

```
#include "armpl.h"

void dlae2_(const double *a, const double *b, const double *c, double *rt1,
            double *rt2);
```

Parameters

A Input parameter.

A is DOUBLE PRECISION

The (1,1) element of the 2-by-2 matrix.

B Input parameter.

B is DOUBLE PRECISION

The (1,2) and (2,1) elements of the 2-by-2 matrix.

C Input parameter.

C is DOUBLE PRECISION

The (2,2) element of the 2-by-2 matrix.

RT1 Output parameter.

RT1 is DOUBLE PRECISION

The eigenvalue of larger absolute value.

RT2 Output parameter.

RT2 is DOUBLE PRECISION

The eigenvalue of smaller absolute value.

Related Information

For this routine in other precisions, please see *slae2*.

4.17.229 dlaebz

dlaebz contains the iteration loops which compute and use the function N(w), which is the count of eigenvalues of a symmetric tridiagonal matrix T less than or equal to its argument w. It performs a choice of two types of loops:

IJOB=1, followed by IJOB=2: It takes as input a list of intervals and returns a list of

```
sufficiently small intervals whose union contains the same
eigenvalues as the union of the original intervals.
The input intervals are (AB(j,1),AB(j,2)], j=1,...,MINP.
The output interval (AB(j,1),AB(j,2)] will contain
eigenvalues NAB(j,1)+1,...,NAB(j,2), where 1 <= j <= MOUT.
```

IJOB=3: It performs a binary search in each input interval

```
(AB(j,1),AB(j,2)] for a point  w(j)  such that
N(w(j))=NVAL(j), and uses  C(j)  as the starting point of
the search.  If such a w(j) is found, then on output
AB(j,1)=AB(j,2)=w.  If no such w(j) is found, then on output
(AB(j,1),AB(j,2)] will be a small interval containing the
point where N(w) jumps through NVAL(j), unless that point
lies outside the initial interval.
```

Note that the intervals are in all cases half-open intervals, i.e., of the form (a,b] , which includes b but not a .

To avoid underflow, the matrix should be scaled so that its largest element is no greater than overflow**(1/2) * underflow**(1/4) in absolute value. To assure the most accurate computation of small eigenvalues, the matrix should be scaled to be not much smaller than that, either.

See W. Kahan "Accurate Eigenvalues of a Symmetric Tridiagonal Matrix", Report CS41, Computer Science Dept., Stanford University, July 21, 1966

Note: the arguments are, in general, *not* checked for unreasonable values.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlaebz(IJOB, NITMAX, N, MMAX, MINP, NBMIN, ABSTOL, RELTOL, PIVMIN,
                  D, E, E2, NVAL, AB, C, MOUT, NAB, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dlaebz_(const armpl_int_t *ijob, const armpl_int_t *nitmax,
             const armpl_int_t *n, const armpl_int_t *mmax,
             const armpl_int_t *minp, const armpl_int_t *nbmin,
             const double *abstol, const double *reltol, const double *pivmin,
             const double *d, const double *e, const double *e2,
             armpl_int_t *nval, double *ab, double *c, armpl_int_t *mout,
             armpl_int_t *nab, double *work, armpl_int_t *iwork,
             armpl_int_t *info);
```

Parameters

IJOB Input parameter.

IJOB is INTEGER

Specifies what is to be done: = 1: Compute NAB for the initial intervals. = 2: Perform bisection iteration to find eigenvalues of T. = 3: Perform bisection iteration to invert N(w), i.e., to find a point which has a specified number of eigenvalues of T to its left. Other values will cause DLAEBZ to return with INFO=-1.

NITMAX Input parameter.

NITMAX is INTEGER

The maximum number of "levels" of bisection to be performed, i.e., an interval of width W will not be made smaller than 2^(-NITMAX) * W. If not all intervals have converged after NITMAX iterations, then INFO is set to the number of non-converged intervals.

N Input parameter.

N is INTEGER

The dimension n of the tridiagonal matrix T. It must be at least 1.

MMAX Input parameter.

MMAX is INTEGER

The maximum number of intervals. If more than MMAX intervals are generated, then DLAEBZ will quit with INFO=MMAX+1.

MINP Input parameter.

MINP is INTEGER

The initial number of intervals. It may not be greater than MMAX.

NBMIN Input parameter.

NBMIN is INTEGER

The smallest number of intervals that should be processed using a vector loop. If zero, then only the scalar loop will be used.

ABSTOL Input parameter.

ABSTOL is DOUBLE PRECISION

The minimum (absolute) width of an interval. When an interval is narrower than ABSTOL, or than RELTOL times the larger (in magnitude) endpoint, then it is considered to be sufficiently small, i.e., converged. This must be at least zero.

RELTOL Input parameter.

RELTOL is DOUBLE PRECISION

The minimum relative width of an interval. When an interval is narrower than ABSTOL, or than RELTOL times the larger (in magnitude) endpoint, then it is considered to be sufficiently small, i.e., converged. Note: this should always be at least radix*machine epsilon.

PIVMIN Input parameter.

PIVMIN is DOUBLE PRECISION

The minimum absolute value of a "pivot" in the Sturm sequence loop. This must be at least max |e(j)**2|*safe_min and at least safe_min, where safe_min is at least the smallest number that can divide one without overflow.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The diagonal elements of the tridiagonal matrix T.

E Input parameter.

E is DOUBLE PRECISION

E is an array, dimension (N). The offdiagonal elements of the tridiagonal matrix T in positions 1 through N-1. E(N) is arbitrary.

E2 Input parameter.

E2 is DOUBLE PRECISION

E2 is an array, dimension (N). The squares of the offdiagonal elements of the tridiagonal matrix T. E2(N) is ignored.

NVAL Input and output parameter.

NVAL is INTEGER array, dimension (MINP)

If IJOB=1 or 2, not referenced. If IJOB=3, the desired values of N(w). The elements of NVAL will be reordered to correspond with the intervals in AB. Thus, NVAL(j) on output will not, in general be the same as NVAL(j) on input, but it will correspond with the interval (AB(j,1),AB(j,2)] on output.

AB Input and output parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (MMAX,2). The endpoints of the intervals. AB(j,1) is a(j), the left endpoint of the j-th interval, and AB(j,2) is b(j), the right endpoint of the j-th interval. The input intervals will, in general, be modified, split, and reordered by the calculation.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (MMAX). If IJOB=1, ignored. If IJOB=2, workspace. If IJOB=3, then on input C(j) should be initialized to the first search point in the binary search.

MOUT Output parameter.

MOUT is INTEGER

If IJOB=1, the number of eigenvalues in the intervals. If IJOB=2 or 3, the number of intervals output. If IJOB=3, MOUT will equal MINP.

NAB Input and output parameter.

NAB is INTEGER array, dimension (MMAX,2)

If IJOB=1, then on output NAB(i,j) will be set to N(AB(i,j)). If IJOB=2, then on input, NAB(i,j) should be set. It must satisfy the condition: N(AB(i,1)) <= NAB(i,1) <= NAB(i,2) <= N(AB(i,2)), which means that in interval i only eigenvalues NAB(i,1)+1,...,NAB(i,2) will be considered. Usually, NAB(i,j)=N(AB(i,j)), from a previous call to DLAEBZ with IJOB=1. On output, NAB(i,j) will contain max(na(k),min(nb(k),N(AB(i,j)))), where k is the index of the input interval that the output interval (AB(j,1),AB(j,2)] came from, and na(k) and nb(k) are the the input values of NAB(k,1) and NAB(k,2). If IJOB=3, then on output, NAB(i,j) contains N(AB(i,j)), unless N(w) > NVAL(i) for all search points w , in which case NAB(i,1) will not be modified, i.e., the output value will be the same as the input value (modulo reorderings – see NVAL and AB), or unless N(w) < NVAL(i) for all search points w , in which case NAB(i,2) will not be modified. Normally, NAB should be set to some distinctive value(s) before DLAEBZ is called.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MMAX). Workspace.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MMAX)

Workspace.

INFO Output parameter.

INFO is INTEGER

= 0: All intervals converged. = 1–MMAX: The last INFO intervals did not converge. = MMAX+1: More than MMAX intervals were generated.

Related Information

For this routine in other precisions, please see *slaebz*.

4.17.230 dlaed0

dlaed0 computes all eigenvalues and corresponding eigenvectors of a symmetric tridiagonal matrix using the divide and conquer method.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlaed0(ICOMPQ, QSIZ, N, D, E, Q, LDQ, QSTORE, LDQS, WORK, IWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dlaed0_(const armpl_int_t *icompq, const armpl_int_t *qsiz,
             const armpl_int_t *n, double *d, const double *e, double *q,
             const armpl_int_t *ldq, double *qstore, const armpl_int_t *ldqs,
             double *work, armpl_int_t *iwork, armpl_int_t *info);
```

Parameters

ICOMPQ Input parameter.

ICOMPQ is INTEGER

= 0: Compute eigenvalues only. = 1: Compute eigenvectors of original dense symmetric matrix also. On entry, Q contains the orthogonal matrix used to reduce the original matrix to tridiagonal form. = 2: Compute eigenvalues and eigenvectors of tridiagonal matrix.

QSIZ Input parameter.

QSIZ is INTEGER

The dimension of the orthogonal matrix used to reduce the full matrix to tridiagonal form. QSIZ >= N if ICOMPQ = 1.

N Input parameter.

N is INTEGER

The dimension of the symmetric tridiagonal matrix. N >= 0.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry, the main diagonal of the tridiagonal matrix. On exit, its eigenvalues.

E Input parameter.

E is DOUBLE PRECISION

E is an array, dimension (N-1). The off-diagonal elements of the tridiagonal matrix. On exit, E has been destroyed.

Q Input and output parameter.

Q is DOUBLE PRECISION

Q is an array, dimension (LDQ, N). On entry, Q must contain an N-by-N orthogonal matrix. If ICOMPQ = 0 Q is not referenced. If ICOMPQ = 1 On entry, Q is a subset of the columns of the orthogonal matrix used to reduce the full matrix to tridiagonal form corresponding to the subset of the full matrix which is being decomposed at this time. If ICOMPQ = 2 On entry, Q will be the identity matrix. On exit, Q contains the eigenvectors of the tridiagonal matrix.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. If eigenvectors are desired, then LDQ >= max(1, N). In any case, LDQ >= 1.

QSTORE Output parameter.

QSTORE is DOUBLE PRECISION

QSTORE is an array, dimension (LDQS, N). Referenced only when ICOMPQ = 1. Used to store parts of the eigenvector matrix when the updating matrix multiplies take place.

LDQS Input parameter.

LDQS is INTEGER

The leading dimension of the array QSTORE. If ICOMPQ = 1, then LDQS >= max(1, N). In any case, LDQS >= 1.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array,. If ICOMPQ = 0 or 1, the dimension of WORK must be at least 1 + 3* N + 2*N*lg N + 3*N**2 (lg(N) = smallest integer k such that 2^k >= N) If ICOMPQ = 2, the dimension of WORK must be at least 4* N + N**2.

IWORK Output parameter.

IWORK is INTEGER array,

If ICOMPQ = 0 or 1, the dimension of IWORK must be at least 6 + 6* N + 5*N*lg N. (lg(N) = smallest integer k such that 2^k >= N) If ICOMPQ = 2, the dimension of IWORK must be at least 3 + 5*N.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: The algorithm failed to compute an eigenvalue while working on the submatrix lying in rows and columns INFO/(N+1) through mod(INFO,N+1).

Related Information

For this routine in other precisions, please see *claed0*, *slaed0* and *zlaed0*.

4.17.231 dlaed1

dlaed1 computes the updated eigensystem of a diagonal matrix after modification by a rank-one symmetric matrix. This routine is used only for the eigenproblem which requires all eigenvalues and eigenvectors of a tridiagonal matrix. DLAED7 handles the case in which eigenvalues only or eigenvalues and eigenvectors of a full symmetric matrix (which was reduced to tridiagonal form) are desired.

```
T = Q(in) ( D(in) + RHO * Z*Z**T ) Q**T(in) = Q(out) * D(out) * Q**T(out)

where Z = Q**T*u, u is a vector of length N with ones in the
CUTPNT and CUTPNT + 1 th elements and zeros elsewhere.

The eigenvectors of the original matrix are stored in Q, and the
eigenvalues are in D.  The algorithm consists of three stages:

   The first stage consists of deflating the size of the problem
   when there are multiple eigenvalues or if there is a zero in
   the Z vector.  For each such occurrence the dimension of the
   secular equation problem is reduced by one.  This stage is
   performed by the routine DLAED2.

   The second stage consists of calculating the updated
   eigenvalues. This is done by finding the roots of the secular
   equation via the routine DLAED4 (as called by DLAED3).
   This routine also calculates the eigenvectors of the current
   problem.

   The final stage consists of computing the updated eigenvectors
   directly using the updated eigenvalues.  The eigenvectors for
   the current problem are multiplied with the eigenvectors from
   the overall problem.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dlaed1(N, D, Q, LDQ, INDXQ, RHO, CUTPNT, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dlaed1_(const armpl_int_t *n, double *d, double *q,
             const armpl_int_t *ldq, armpl_int_t *indxq, const double *rho,
             const armpl_int_t *cutpnt, double *work, armpl_int_t *iwork,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The dimension of the symmetric tridiagonal matrix. N >= 0.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry, the eigenvalues of the rank-1-perturbed matrix. On exit, the eigenvalues of the repaired matrix.

Q Input and output parameter.

Q is DOUBLE PRECISION

Q is an array, dimension (LDQ, N). On entry, the eigenvectors of the rank-1-perturbed matrix. On exit, the eigenvectors of the repaired tridiagonal matrix.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, N).

INDXQ Input and output parameter.

INDXQ is INTEGER array, dimension (N)

On entry, the permutation which separately sorts the two subproblems in D into ascending order. On exit, the permutation which will reintegrate the subproblems back into sorted order, i.e. D(INDXQ(I = 1, N)) will be in ascending order.

RHO Input parameter.

RHO is DOUBLE PRECISION

The subdiagonal entry used to create the rank-1 modification.

CUTPNT Input parameter.

CUTPNT is INTEGER

The location of the last eigenvalue in the leading sub-matrix. min(1, N) <= CUTPNT <= N/2.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (4* N + N2)** .

IWORK Output parameter.

IWORK is INTEGER array, dimension (4*N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = 1, an eigenvalue did not converge

Related Information

For this routine in other precisions, please see *slaed1*.

4.17.232 dlaed2

dlaed2 merges the two sets of eigenvalues together into a single sorted set. Then it tries to deflate the size of the problem. There are two ways in which deflation can occur: when two or more eigenvalues are close together or if there is a tiny entry in the Z vector. For each such occurrence the order of the related secular equation problem is reduced by one.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlaed2(K, N, N1, D, Q, LDQ, INDXQ, RHO, Z, DLAMDA, W, Q2, INDX,
                  INDXC, INDXP, COLTYP, INFO)
```

C specification:

```
#include "armpl.h"

void dlaed2_(armpl_int_t *k, const armpl_int_t *n, const armpl_int_t *n1,
             double *d, double *q, const armpl_int_t *ldq, armpl_int_t *indxq,
             double *rho, const double *z, double *dlamda, double *w,
             double *q2, armpl_int_t *indx, armpl_int_t *indxc,
             armpl_int_t *indxp, armpl_int_t *coltyp, armpl_int_t *info);
```

Parameters

K Output parameter.

K is INTEGER

The number of non-deflated eigenvalues, and the order of the related secular equation. 0 <= K <=N.

N Input parameter.

N is INTEGER

The dimension of the symmetric tridiagonal matrix. N >= 0.

N1 Input parameter.

N1 is INTEGER

The location of the last eigenvalue in the leading sub-matrix. min(1, N) <= N1 <= N/2.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry, D contains the eigenvalues of the two submatrices to be combined. On exit, D contains the trailing (N-K) updated eigenvalues (those which were deflated) sorted into increasing order.

Q Input and output parameter.

Q is DOUBLE PRECISION

Q is an array, dimension (LDQ, N). On entry, Q contains the eigenvectors of two submatrices in the two square blocks with corners at (1,1), (N1, N1) and (N1+1, N1+1), (N, N). On exit, Q contains the trailing (N-K) updated eigenvectors (those which were deflated) in its last N-K columns.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, N).

INDXQ Input and output parameter.

INDXQ is INTEGER array, dimension (N)

The permutation which separately sorts the two sub-problems in D into ascending order. Note that elements in the second half of this permutation must first have N1 added to their values. Destroyed on exit.

RHO Input and output parameter.

RHO is DOUBLE PRECISION

On entry, the off-diagonal element associated with the rank-1 cut which originally split the two submatrices which are now being recombined. On exit, RHO has been modified to the value required by DLAED3.

Z Input parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (N). On entry, Z contains the updating vector (the last row of the first sub-eigenvector matrix and the first row of the second sub-eigenvector matrix). On exit, the contents of Z have been destroyed by the updating process.

DLAMDA Output parameter.

DLAMDA is DOUBLE PRECISION

DLAMDA is an array, dimension (N). A copy of the first K eigenvalues which will be used by DLAED3 to form the secular equation.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). The first k values of the final deflation-altered z-vector which will be passed to DLAED3.

Q2 Output parameter.

Q2 is DOUBLE PRECISION

Q2 is an array, dimension (N1**2+(N-N1)**2). A copy of the first K eigenvectors which will be used by DLAED3 in a matrix multiply (DGEMM) to solve for the new eigenvectors.

INDX Output parameter.

INDX is INTEGER array, dimension (N)

The permutation used to sort the contents of DLAMDA into ascending order.

INDXC Output parameter.

INDXC is INTEGER array, dimension (N)

The permutation used to arrange the columns of the deflated Q matrix into three groups: the first group contains non-zero elements only at and above N1, the second contains non-zero elements only below N1, and the third is dense.

INDXP Output parameter.

INDXP is INTEGER array, dimension (N)

The permutation used to place deflated values of D at the end of the array. INDXP(1:K) points to the nondeflated D-values and INDXP(K+1:N) points to the deflated eigenvalues.

COLTYP Output parameter.

COLTYP is INTEGER array, dimension (N)

During execution, a label which will indicate which of the following types a column in the Q2 matrix is: 1 : non-zero in the upper half only; 2 : dense; 3 : non-zero in the lower half only; 4 : deflated. On exit, COLTYP(i) is the number of columns of type i, for i=1 to 4 only.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *slaed2*.

4.17.233 dlaed3

dlaed3 finds the roots of the secular equation, as defined by the values in D, W, and RHO, between 1 and K. It makes the appropriate calls to DLAED4 and then updates the eigenvectors by multiplying the matrix of eigenvectors of the pair of eigensystems being combined by the matrix of eigenvectors of the K-by-K system which is solved here.

This code makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlaed3(K, N, N1, D, Q, LDQ, RHO, DLAMDA, Q2, INDX, CTOT, W, S,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dlaed3_(const armpl_int_t *k, const armpl_int_t *n,
             const armpl_int_t *n1, double *d, double *q,
             const armpl_int_t *ldq, const double *rho, double *dlamda,
             const double *q2, const armpl_int_t *indx,
             const armpl_int_t *ctot, double *w, double *s,
             armpl_int_t *info);
```

Parameters

K Input parameter.

K is INTEGER

The number of terms in the rational function to be solved by DLAED4. K >= 0.

N Input parameter.

N is INTEGER

The number of rows and columns in the Q matrix. N >= K (deflation may result in N>K).

N1 Input parameter.

N1 is INTEGER

The location of the last eigenvalue in the leading submatrix. min(1, N) <= N1 <= N/2.

D Output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). D(I) contains the updated eigenvalues for 1 <= I <= K.

Q Output parameter.

Q is DOUBLE PRECISION

Q is an array, dimension (LDQ, N). Initially the first K columns are used as workspace. On output the columns 1 to K contain the updated eigenvectors.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, N).

RHO Input parameter.

RHO is DOUBLE PRECISION

The value of the parameter in the rank one update equation. RHO >= 0 required.

DLAMDA Input and output parameter.

DLAMDA is DOUBLE PRECISION

DLAMDA is an array, dimension (K). The first K elements of this array contain the old roots of the deflated updating problem. These are the poles of the secular equation. May be changed on output by having lowest order bit set to zero on Cray X-MP, Cray Y-MP, Cray-2, or Cray C-90, as described above.

Q2 Input parameter.

Q2 is DOUBLE PRECISION

Q2 is an array, dimension (LDQ2*N). The first K columns of this matrix contain the non-deflated eigenvectors for the split problem.

INDX Input parameter.

INDX is INTEGER array, dimension (N)

The permutation used to arrange the columns of the deflated Q matrix into three groups (see DLAED2). The rows of the eigenvectors found by DLAED4 must be likewise permuted before the matrix multiply can take place.

CTOT Input parameter.

CTOT is INTEGER array, dimension (4)

A count of the total number of the various types of columns in Q, as described in INDX. The fourth column type is any column which has been deflated.

W Input and output parameter.

W is DOUBLE PRECISION

W is an array, dimension (K). The first K elements of this array contain the components of the deflation-adjusted updating vector. Destroyed on output.

S Output parameter.

S is DOUBLE PRECISION

S is an array, dimension (N1 + 1)*K. Will contain the eigenvectors of the repaired matrix which will be multiplied by the previously accumulated eigenvectors to update the system.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = 1, an eigenvalue did not converge

Related Information

For this routine in other precisions, please see *slaed3*.

4.17.234 dlaed4

This subroutine computes the I-th updated eigenvalue of a symmetric rank-one modification to a diagonal matrix whose elements are given in the array d, and that

```
D(i) < D(j)  for  i < j
```

and that RHO > 0. This is arranged by the calling routine, and is no loss in generality. The rank-one modified system is thus

```
diag( D )  +  RHO * Z * Z_transpose.
```

where we assume the Euclidean norm of Z is 1.

The method consists of approximating the rational functions in the secular equation by simpler interpolating rational functions.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlaed4(N, I, D, Z, DELTA, RHO, DLAM, INFO)
```

C specification:

```
#include "armpl.h"

void dlaed4_(const armpl_int_t *n, const armpl_int_t *i, const double *d,
             const double *z, double *delta, const double *rho, double *dlam,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The length of all arrays.

I Input parameter.

I is INTEGER

The index of the eigenvalue to be computed. 1 <= I <= N.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The original eigenvalues. It is assumed that they are in order, D(I) < D(J) for I < J.

Z Input parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (N). The components of the updating vector.

DELTA Output parameter.

DELTA is DOUBLE PRECISION

DELTA is an array, dimension (N). If N .GT. 2, DELTA contains (D(j) - lambda_I) in its j-th component. If N = 1, then DELTA(1) = 1. If N = 2, see DLAED5 for detail. The vector DELTA contains the information necessary to construct the eigenvectors by DLAED3 and DLAED9.

RHO Input parameter.

RHO is DOUBLE PRECISION

The scalar in the symmetric updating formula.

DLAM Output parameter.

DLAM is DOUBLE PRECISION

The computed lambda_I, the I-th updated eigenvalue.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit > 0: if INFO = 1, the updating process failed.

Related Information

For this routine in other precisions, please see *slaed4*.

4.17.235 dlaed5

This subroutine computes the I-th eigenvalue of a symmetric rank-one modification of a 2-by-2 diagonal matrix

```
diag( D )  +  RHO * Z * transpose(Z) .
```

The diagonal elements in the array D are assumed to satisfy

```
D(i) < D(j)  for  i < j .
```

We also assume RHO > 0 and that the Euclidean norm of the vector Z is one.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlaed5(I, D, Z, DELTA, RHO, DLAM)
```

C specification:

```
#include "armpl.h"

void dlaed5_(const armpl_int_t *i, const double *d, const double *z,
             double *delta, const double *rho, double *dlam);
```

Parameters

I Input parameter.

I is INTEGER

The index of the eigenvalue to be computed. I = 1 or I = 2.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (2). The original eigenvalues. We assume D(1) < D(2).

Z Input parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (2). The components of the updating vector.

DELTA Output parameter.

DELTA is DOUBLE PRECISION

DELTA is an array, dimension (2). The vector DELTA contains the information necessary to construct the eigenvectors.

RHO Input parameter.

RHO is DOUBLE PRECISION

The scalar in the symmetric updating formula.

DLAM Output parameter.

DLAM is DOUBLE PRECISION

The computed lambda_I, the I-th updated eigenvalue.

Related Information

For this routine in other precisions, please see *slaed5*.

4.17.236 dlaed6

dlaed6 computes the positive or negative root (closest to the origin) of

```
z(1)          z(2)          z(3)
```

f(x) = rho + ——— + ——— + ———

```
d(1)-x      d(2)-x      d(3)-x
```

It is assumed that

```
if ORGATI = .true. the root is between d(2) and d(3);
otherwise it is between d(1) and d(2)
```

This routine will be called by DLAED4 when necessary. In most cases, the root sought is the smallest in magnitude, though it might not be in some extremely rare situations.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlaed6(KNITER, ORGATI, RHO, D, Z, FINIT, TAU, INFO)
```

C specification:

```
#include "armpl.h"

void dlaed6_(const armpl_int_t *kniter, const armpl_int_t *orgati,
             const double *rho, const double *d, const double *z,
             const double *finit, double *tau, armpl_int_t *info);
```

Parameters

KNITER Input parameter.

KNITER is INTEGER

Refer to DLAED4 for its significance.

ORGATI Input parameter.

ORGATI is LOGICAL

If ORGATI is true, the needed root is between d(2) and d(3); otherwise it is between d(1) and d(2). See DLAED4 for further details.

RHO Input parameter.

RHO is DOUBLE PRECISION

Refer to the equation f(x) above.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (3). D satisfies d(1) < d(2) < d(3).

Z Input parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (3). Each of the elements in z must be positive.

FINIT Input parameter.

FINIT is DOUBLE PRECISION

The value of f at 0. It is more accurate than the one evaluated inside this routine (if someone wants to do so).

TAU Output parameter.

TAU is DOUBLE PRECISION

The root of the equation f(x).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit > 0: if INFO = 1, failure to converge

Related Information

For this routine in other precisions, please see *slaed6*.

4.17.237 dlaed7

dlaed7 computes the updated eigensystem of a diagonal matrix after modification by a rank-one symmetric matrix. This routine is used only for the eigenproblem which requires all eigenvalues and optionally eigenvectors of a dense symmetric matrix that has been reduced to tridiagonal form. DLAED1 handles the case in which all eigenvalues and eigenvectors of a symmetric tridiagonal matrix are desired.

```
T = Q(in) ( D(in) + RHO * Z*Z**T ) Q**T(in) = Q(out) * D(out) * Q**T(out)

where Z = Q**Tu, u is a vector of length N with ones in the
CUTPNT and CUTPNT + 1 th elements and zeros elsewhere.

The eigenvectors of the original matrix are stored in Q, and the
eigenvalues are in D.  The algorithm consists of three stages:

   The first stage consists of deflating the size of the problem
   when there are multiple eigenvalues or if there is a zero in
   the Z vector.  For each such occurrence the dimension of the
   secular equation problem is reduced by one.  This stage is
   performed by the routine DLAED8.

   The second stage consists of calculating the updated
   eigenvalues. This is done by finding the roots of the secular
   equation via the routine DLAED4 (as called by DLAED9).
   This routine also calculates the eigenvectors of the current
   problem.

   The final stage consists of computing the updated eigenvectors
   directly using the updated eigenvalues.  The eigenvectors for
   the current problem are multiplied with the eigenvectors from
   the overall problem.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dlaed7(ICOMPQ, N, QSIZ, TLVLS, CURLVL, CURPBM, D, Q, LDQ, INDXQ,
                  RHO, CUTPNT, QSTORE, QPTR, PRMPTR, PERM, GIVPTR, GIVCOL,
                  GIVNUM, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dlaed7_(const armpl_int_t *icompq, const armpl_int_t *n,
             const armpl_int_t *qsiz, const armpl_int_t *tlvls,
             const armpl_int_t *curlvl, const armpl_int_t *curpbm, double *d,
             double *q, const armpl_int_t *ldq, armpl_int_t *indxq,
             const double *rho, const armpl_int_t *cutpnt, double *qstore,
             armpl_int_t *qptr, const armpl_int_t *prmptr,
             const armpl_int_t *perm, const armpl_int_t *givptr,
             const armpl_int_t *givcol, const double *givnum, double *work,
             armpl_int_t *iwork, armpl_int_t *info);
```

Parameters

ICOMPQ Input parameter.

ICOMPQ is INTEGER

= 0: Compute eigenvalues only. = 1: Compute eigenvectors of original dense symmetric matrix also. On entry, Q contains the orthogonal matrix used to reduce the original matrix to tridiagonal form.

N Input parameter.

N is INTEGER

The dimension of the symmetric tridiagonal matrix. N >= 0.

QSIZ Input parameter.

QSIZ is INTEGER

The dimension of the orthogonal matrix used to reduce the full matrix to tridiagonal form. QSIZ >= N if ICOMPQ = 1.

TLVLS Input parameter.

TLVLS is INTEGER

The total number of merging levels in the overall divide and conquer tree.

CURLVL Input parameter.

CURLVL is INTEGER

The current level in the overall merge routine, 0 <= CURLVL <= TLVLS.

CURPBM Input parameter.

CURPBM is INTEGER

The current problem in the current level in the overall merge routine (counting from upper left to lower right).

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry, the eigenvalues of the rank-1-perturbed matrix. On exit, the eigenvalues of the repaired matrix.

Q Input and output parameter.

Q is DOUBLE PRECISION

Q is an array, dimension (LDQ, N). On entry, the eigenvectors of the rank-1-perturbed matrix. On exit, the eigenvectors of the repaired tridiagonal matrix.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, N).

INDXQ Output parameter.

INDXQ is INTEGER array, dimension (N)

The permutation which will reintegrate the subproblem just solved back into sorted order, i.e., D(INDXQ(I = 1, N)) will be in ascending order.

RHO Input parameter.

RHO is DOUBLE PRECISION

The subdiagonal element used to create the rank-1 modification.

CUTPNT Input parameter.

CUTPNT is INTEGER

Contains the location of the last eigenvalue in the leading sub-matrix. min(1, N) <= CUTPNT <= N.

QSTORE Input and output parameter.

QSTORE is DOUBLE PRECISION

QSTORE is an array, dimension (N**2+1). Stores eigenvectors of submatrices encountered during divide and conquer, packed together. QPTR points to beginning of the submatrices.

QPTR Input and output parameter.

QPTR is INTEGER array, dimension (N+2)

List of indices pointing to beginning of submatrices stored in QSTORE. The submatrices are numbered starting at the bottom left of the divide and conquer tree, from left to right and bottom to top.

PRMPTR Input parameter.

PRMPTR is INTEGER array, dimension (N lg N)

Contains a list of pointers which indicate where in PERM a level's permutation is stored. PRMPTR(i+1) - PRMPTR(i) indicates the size of the permutation and also the size of the full, non-deflated problem.

PERM Input parameter.

PERM is INTEGER array, dimension (N lg N)

Contains the permutations (from deflation and sorting) to be applied to each eigenblock.

GIVPTR Input parameter.

GIVPTR is INTEGER array, dimension (N lg N)

Contains a list of pointers which indicate where in GIVCOL a level's Givens rotations are stored. GIVPTR(i+1) - GIVPTR(i) indicates the number of Givens rotations.

GIVCOL Input parameter.

GIVCOL is INTEGER array, dimension (2, N lg N)

Each pair of numbers indicates a pair of columns to take place in a Givens rotation.

GIVNUM Input parameter.

GIVNUM is DOUBLE PRECISION

GIVNUM is an array, dimension (2, N lg N). Each number indicates the S value to be used in the corresponding Givens rotation.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (3*N+2*QSIZ*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (4*N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = 1, an eigenvalue did not converge

Related Information

For this routine in other precisions, please see *claed7*, *slaed7* and *zlaed7*.

4.17.238 dlaed8

dlaed8 merges the two sets of eigenvalues together into a single sorted set. Then it tries to deflate the size of the problem. There are two ways in which deflation can occur: when two or more eigenvalues are close together or if there is a tiny element in the Z vector. For each such occurrence the order of the related secular equation problem is reduced by one.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlaed8(ICOMPQ, K, N, QSIZ, D, Q, LDQ, INDXQ, RHO, CUTPNT, Z,
                  DLAMDA, Q2, LDQ2, W, PERM, GIVPTR, GIVCOL, GIVNUM, INDXP,
                  INDX, INFO)
```

C specification:

```
#include "armpl.h"

void dlaed8_(const armpl_int_t *icompq, armpl_int_t *k, const armpl_int_t *n,
             const armpl_int_t *qsiz, double *d, double *q,
             const armpl_int_t *ldq, const armpl_int_t *indxq, double *rho,
             const armpl_int_t *cutpnt, const double *z, double *dlamda,
             double *q2, const armpl_int_t *ldq2, double *w,
             armpl_int_t *perm, armpl_int_t *givptr, armpl_int_t *givcol,
             double *givnum, armpl_int_t *indxp, armpl_int_t *indx,
             armpl_int_t *info);
```

Parameters

ICOMPQ Input parameter.

ICOMPQ is INTEGER

= 0: Compute eigenvalues only. = 1: Compute eigenvectors of original dense symmetric matrix also. On entry, Q contains the orthogonal matrix used to reduce the original matrix to tridiagonal form.

K Output parameter.

K is INTEGER

The number of non-deflated eigenvalues, and the order of the related secular equation.

N Input parameter.

N is INTEGER

The dimension of the symmetric tridiagonal matrix. N >= 0.

QSIZ Input parameter.

QSIZ is INTEGER

The dimension of the orthogonal matrix used to reduce the full matrix to tridiagonal form. QSIZ >= N if ICOMPQ = 1.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry, the eigenvalues of the two submatrices to be combined. On exit, the trailing (N-K) updated eigenvalues (those which were deflated) sorted into increasing order.

Q Input and output parameter.

Q is DOUBLE PRECISION

Q is an array, dimension (LDQ, N). If ICOMPQ = 0, Q is not referenced. Otherwise, on entry, Q contains the eigenvectors of the partially solved system which has been previously updated in matrix multiplies with other partially solved eigensystems. On exit, Q contains the trailing (N-K) updated eigenvectors (those which were deflated) in its last N-K columns.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, N).

INDXQ Input parameter.

INDXQ is INTEGER array, dimension (N)

The permutation which separately sorts the two sub-problems in D into ascending order. Note that elements in the second half of this permutation must first have CUTPNT added to their values in order to be accurate.

RHO Input and output parameter.

RHO is DOUBLE PRECISION

On entry, the off-diagonal element associated with the rank-1 cut which originally split the two submatrices which are now being recombined. On exit, RHO has been modified to the value required by DLAED3.

CUTPNT Input parameter.

CUTPNT is INTEGER

The location of the last eigenvalue in the leading sub-matrix. min(1, N) <= CUTPNT <= N.

Z Input parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (N). On entry, Z contains the updating vector (the last row of the first sub-eigenvector matrix and the first row of the second sub-eigenvector matrix). On exit, the contents of Z are destroyed by the updating process.

DLAMDA Output parameter.

DLAMDA is DOUBLE PRECISION

DLAMDA is an array, dimension (N). A copy of the first K eigenvalues which will be used by DLAED3 to form the secular equation.

Q2 Output parameter.

Q2 is DOUBLE PRECISION

Q2 is an array, dimension (LDQ2, N). If ICOMPQ = 0, Q2 is not referenced. Otherwise, a copy of the first K eigenvectors which will be used by DLAED7 in a matrix multiply (DGEMM) to update the new eigenvectors.

LDQ2 Input parameter.

LDQ2 is INTEGER

The leading dimension of the array Q2. LDQ2 >= max(1, N).

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). The first k values of the final deflation-altered z-vector and will be passed to DLAED3.

PERM Output parameter.

PERM is INTEGER array, dimension (N)

The permutations (from deflation and sorting) to be applied to each eigenblock.

GIVPTR Output parameter.

GIVPTR is INTEGER

The number of Givens rotations which took place in this subproblem.

GIVCOL Output parameter.

GIVCOL is INTEGER array, dimension (2, N)

Each pair of numbers indicates a pair of columns to take place in a Givens rotation.

GIVNUM Output parameter.

GIVNUM is DOUBLE PRECISION

GIVNUM is an array, dimension (2, N). Each number indicates the S value to be used in the corresponding Givens rotation.

INDXP Output parameter.

INDXP is INTEGER array, dimension (N)

The permutation used to place deflated values of D at the end of the array. INDXP(1:K) points to the nondeflated D-values and INDXP(K+1:N) points to the deflated eigenvalues.

INDX Output parameter.

INDX is INTEGER array, dimension (N)

The permutation used to sort the contents of D into ascending order.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *claed8*, *slaed8* and *zlaed8*.

4.17.239 dlaed9

dlaed9 finds the roots of the secular equation, as defined by the values in D, Z, and RHO, between KSTART and KSTOP. It makes the appropriate calls to DLAED4 and then stores the new matrix of eigenvectors for use in calculating the next level of Z vectors.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlaed9(K, KSTART, KSTOP, N, D, Q, LDQ, RHO, DLAMDA, W, S, LDS,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dlaed9_(const armpl_int_t *k, const armpl_int_t *kstart,
             const armpl_int_t *kstop, const armpl_int_t *n, double *d,
             double *q, const armpl_int_t *ldq, const double *rho,
             const double *dlamda, const double *w, double *s,
             const armpl_int_t *lds, armpl_int_t *info);
```

Parameters

K Input parameter.

K is INTEGER

The number of terms in the rational function to be solved by DLAED4. K >= 0.

KSTART Input parameter.

KSTART is INTEGER

KSTOP Input parameter.

KSTOP is INTEGER

The updated eigenvalues Lambda(I), KSTART <= I <= KSTOP are to be computed. 1 <= KSTART <= KSTOP <= K.

N Input parameter.

N is INTEGER

The number of rows and columns in the Q matrix. N >= K (delation may result in N > K).

D Output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). D(I) contains the updated eigenvalues for KSTART <= I <= KSTOP.

Q Output parameter.

Q is DOUBLE PRECISION

Q is an array, dimension (LDQ, N) .

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, N).

RHO Input parameter.

RHO is DOUBLE PRECISION

The value of the parameter in the rank one update equation. RHO >= 0 required.

DLAMDA Input parameter.

DLAMDA is DOUBLE PRECISION

DLAMDA is an array, dimension (K). The first K elements of this array contain the old roots of the deflated updating problem. These are the poles of the secular equation.

W Input parameter.

W is DOUBLE PRECISION

W is an array, dimension (K). The first K elements of this array contain the components of the deflation-adjusted updating vector.

S Output parameter.

S is DOUBLE PRECISION

S is an array, dimension (LDS, K). Will contain the eigenvectors of the repaired matrix which will be stored for subsequent Z vector calculation and multiplied by the previously accumulated eigenvectors to update the system.

LDS Input parameter.

LDS is INTEGER

The leading dimension of S. LDS >= max(1, K).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = 1, an eigenvalue did not converge

Related Information

For this routine in other precisions, please see *slaed9*.

4.17.240 dlaeda

dlaeda computes the Z vector corresponding to the merge step in the CURLVLth step of the merge process with TLVLS steps for the CURPBMth problem.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlaeda(N, TLVLS, CURLVL, CURPBM, PRMPTR, PERM, GIVPTR, GIVCOL,
                  GIVNUM, Q, QPTR, Z, ZTEMP, INFO)
```

C specification:

```
#include "armpl.h"

void dlaeda_(const armpl_int_t *n, const armpl_int_t *tlvls,
             const armpl_int_t *curlvl, const armpl_int_t *curpbm,
             const armpl_int_t *prmptr, const armpl_int_t *perm,
             const armpl_int_t *givptr, const armpl_int_t *givcol,
             const double *givnum, const double *q, const armpl_int_t *qptr,
             double *z, double *ztemp, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The dimension of the symmetric tridiagonal matrix. N >= 0.

TLVLS Input parameter.

TLVLS is INTEGER

The total number of merging levels in the overall divide and conquer tree.

CURLVL Input parameter.

CURLVL is INTEGER

The current level in the overall merge routine, 0 <= curlvl <= tlvls.

CURPBM Input parameter.

CURPBM is INTEGER

The current problem in the current level in the overall merge routine (counting from upper left to lower right).

PRMPTR Input parameter.

PRMPTR is INTEGER array, dimension (N lg N)

Contains a list of pointers which indicate where in PERM a level's permutation is stored. PRMPTR(i+1) - PRMPTR(i) indicates the size of the permutation and incidentally the size of the full, non-deflated problem.

PERM Input parameter.

PERM is INTEGER array, dimension (N lg N)

Contains the permutations (from deflation and sorting) to be applied to each eigenblock.

GIVPTR Input parameter.

GIVPTR is INTEGER array, dimension (N lg N)

Contains a list of pointers which indicate where in GIVCOL a level's Givens rotations are stored. GIVPTR(i+1) - GIVPTR(i) indicates the number of Givens rotations.

GIVCOL Input parameter.

GIVCOL is INTEGER array, dimension (2, N lg N)

Each pair of numbers indicates a pair of columns to take place in a Givens rotation.

GIVNUM Input parameter.

GIVNUM is DOUBLE PRECISION

GIVNUM is an array, dimension (2, N lg N). Each number indicates the S value to be used in the corresponding Givens rotation.

Q Input parameter.

Q is DOUBLE PRECISION

Q is an array, dimension (N**2). Contains the square eigenblocks from previous levels, the starting positions for blocks are given by QPTR.

QPTR Input parameter.

QPTR is INTEGER array, dimension (N+2)

Contains a list of pointers which indicate where in Q an eigenblock is stored. SQRT(QPTR(i+1) - QPTR(i)) indicates the size of the block.

Z Output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (N). On output this vector contains the updating vector (the last row of the first sub-eigenvector matrix and the first row of the second sub-eigenvector matrix).

ZTEMP Output parameter.

ZTEMP is DOUBLE PRECISION

ZTEMP is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *slaeda*.

4.17.241 dlaein

dlaein uses inverse iteration to find a right or left eigenvector corresponding to the eigenvalue (WR,WI) of a real upper Hessenberg matrix H.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlaein(RIGHTV, NOINIT, N, H, LDH, WR, WI, VR, VI, B, LDB, WORK,
                  EPS3, SMLNUM, BIGNUM, INFO)
```

C specification:

```
#include "armpl.h"

void dlaein_(const armpl_int_t *rightv, const armpl_int_t *noinit,
             const armpl_int_t *n, const double *h, const armpl_int_t *ldh,
             const double *wr, const double *wi, double *vr, double *vi,
             double *b, const armpl_int_t *ldb, double *work,
             const double *eps3, const double *smlnum, const double *bignum,
             armpl_int_t *info);
```

Parameters

RIGHTV Input parameter.

RIGHTV is LOGICAL

= .TRUE. : compute right eigenvector; = .FALSE.: compute left eigenvector.

NOINIT Input parameter.

NOINIT is LOGICAL

= .TRUE. : no initial vector supplied in (VR, VI). = .FALSE.: initial vector supplied in (VR, VI).

N Input parameter.

N is INTEGER

The order of the matrix H. N >= 0.

H Input parameter.

H is DOUBLE PRECISION

H is an array, dimension (LDH, N). The upper Hessenberg matrix H.

LDH Input parameter.

LDH is INTEGER

The leading dimension of the array H. LDH >= max(1, N).

WR Input parameter.

WR is DOUBLE PRECISION

WI Input parameter.

WI is DOUBLE PRECISION

The real and imaginary parts of the eigenvalue of H whose corresponding right or left eigenvector is to be computed.

VR Input and output parameter.

VR is DOUBLE PRECISION

VR is an array, dimension (N) .

VI Input and output parameter.

VI is DOUBLE PRECISION

VI is an array, dimension (N). On entry, if NOINIT = .FALSE. and WI = 0.0, VR must contain a real starting vector for inverse iteration using the real eigenvalue WR; if NOINIT = .FALSE. and WI.ne.0.0, VR and VI must contain the real and imaginary parts of a complex starting vector for inverse iteration using the complex eigenvalue (WR, WI); otherwise VR and VI need not be set. On exit, if WI = 0.0 (real eigenvalue), VR contains the computed real eigenvector; if WI.ne.0.0 (complex eigenvalue), VR and VI contain the real and imaginary parts of the computed complex eigenvector. The eigenvector is normalized so that the component of largest magnitude has magnitude 1; here the magnitude of a complex number (x,y) is taken to be |x| + |y|. VI is not referenced if WI = 0.0.

B Output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N) .

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= N+1.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (N) .

EPS3 Input parameter.

EPS3 is DOUBLE PRECISION

A small machine-dependent value which is used to perturb close eigenvalues, and to replace zero pivots.

SMLNUM Input parameter.

SMLNUM is DOUBLE PRECISION

A machine-dependent value close to the underflow threshold.

BIGNUM Input parameter.

BIGNUM is DOUBLE PRECISION

A machine-dependent value close to the overflow threshold.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit = 1: inverse iteration did not converge; VR is set to the last iterate, and so is VI if WI.ne.0.0.

Related Information

For this routine in other precisions, please see *claein*, *slaein* and *zlaein*.

4.17.242 dlaev2

dlaev2 computes the eigendecomposition of a 2-by-2 symmetric matrix

```
[  A   B  ]
[  B   C  ].
```

On return, RT1 is the eigenvalue of larger absolute value, RT2 is the eigenvalue of smaller absolute value, and (CS1,SN1) is the unit right eigenvector for RT1, giving the decomposition

```
[ CS1  SN1 ] [  A   B  ] [ CS1 -SN1 ]  =  [ RT1  0  ]
[-SN1  CS1 ] [  B   C  ] [ SN1  CS1 ]     [  0  RT2 ].
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dlaev2(A, B, C, RT1, RT2, CS1, SN1)
```

C specification:

```
#include "armpl.h"

void dlaev2_(const double *a, const double *b, const double *c, double *rt1,
             double *rt2, double *cs1, double *sn1);
```

Parameters

A Input parameter.

A is DOUBLE PRECISION

The (1,1) element of the 2-by-2 matrix.

B Input parameter.

B is DOUBLE PRECISION

The (1,2) element and the conjugate of the (2,1) element of the 2-by-2 matrix.

C Input parameter.

C is DOUBLE PRECISION

The (2,2) element of the 2-by-2 matrix.

RT1 Output parameter.

RT1 is DOUBLE PRECISION

The eigenvalue of larger absolute value.

RT2 Output parameter.

RT2 is DOUBLE PRECISION

The eigenvalue of smaller absolute value.

CS1 Output parameter.

CS1 is DOUBLE PRECISION

SN1 Output parameter.

SN1 is DOUBLE PRECISION

The vector (CS1, SN1) is a unit right eigenvector for RT1.

Related Information

For this routine in other precisions, please see *claev2*, *slaev2* and *zlaev2*.

4.17.243 dlaexc

dlaexc swaps adjacent diagonal blocks T11 and T22 of order 1 or 2 in an upper quasi-triangular matrix T by an orthogonal similarity transformation.

T must be in Schur canonical form, that is, block upper triangular with 1-by-1 and 2-by-2 diagonal blocks; each 2-by-2 diagonal block has its diagonal elemnts equal and its off-diagonal elements of opposite sign.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlaexc(WANTQ, N, T, LDT, Q, LDQ, J1, N1, N2, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dlaexc_(const armpl_int_t *wantq, const armpl_int_t *n, double *t,
             const armpl_int_t *ldt, double *q, const armpl_int_t *ldq,
             const armpl_int_t *j1, const armpl_int_t *n1,
             const armpl_int_t *n2, double *work, armpl_int_t *info);
```

Parameters

WANTQ Input parameter.

WANTQ is LOGICAL

= .TRUE. : accumulate the transformation in the matrix Q; = .FALSE.: do not accumulate the transformation.

N Input parameter.

N is INTEGER

The order of the matrix T. N >= 0.

T Input and output parameter.

T is DOUBLE PRECISION

T is an array, dimension (LDT, N). On entry, the upper quasi-triangular matrix T, in Schur canonical form. On exit, the updated matrix T, again in Schur canonical form.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

Q Input and output parameter.

Q is DOUBLE PRECISION

Q is an array, dimension (LDQ, N). On entry, if WANTQ is .TRUE., the orthogonal matrix Q. On exit, if WANTQ is .TRUE., the updated matrix Q. If WANTQ is .FALSE., Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= 1; and if WANTQ is .TRUE., LDQ >= N.

J1 Input parameter.

J1 is INTEGER

The index of the first row of the first block T11.

N1 Input parameter.

N1 is INTEGER

The order of the first block T11. N1 = 0, 1 or 2.

N2 Input parameter.

N2 is INTEGER

The order of the second block T22. N2 = 0, 1 or 2.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit = 1: the transformed matrix T would be too far from Schur form; the blocks are not swapped and T and Q are unchanged.

Related Information

For this routine in other precisions, please see *slaexc*.

4.17.244 dlag2

`dlag2` computes the eigenvalues of a 2 x 2 generalized eigenvalue problem A - w B, with scaling as necessary to avoid over-/underflow.

The scaling factor "s" results in a modified eigenvalue equation

```
s A - w B
```

where s is a non-negative scaling factor chosen so that w, w B, and s A do not overflow and, if possible, do not underflow, either.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlag2(A, LDA, B, LDB, SAFMIN, SCALE1, SCALE2, WR1, WR2, WI)
```

C specification:

```
#include "armpl.h"

void dlag2_(const double *a, const armpl_int_t *lda, const double *b,
            const armpl_int_t *ldb, const double *safmin, double *scale1,
            double *scale2, double *wr1, double *wr2, double *wi);
```

Parameters

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, 2). On entry, the 2 x 2 matrix A. It is assumed that its 1-norm is less than 1/SAFMIN. Entries less than sqrt(SAFMIN)*norm(A) are subject to being treated as zero.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= 2.

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, 2). On entry, the 2 x 2 upper triangular matrix B. It is assumed that the one-norm of B is less than 1/SAFMIN. The diagonals should be at least sqrt(SAFMIN) times the largest element of B (in absolute value); if a diagonal is smaller than that, then +/- sqrt(SAFMIN) will be used instead of that diagonal.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= 2.

SAFMIN Input parameter.

SAFMIN is DOUBLE PRECISION

The smallest positive number s.t. 1/SAFMIN does not overflow. (This should always be DLAMCH('S') – it is an argument in order to avoid having to call DLAMCH frequently.)

SCALE1 Output parameter.

SCALE1 is DOUBLE PRECISION

A scaling factor used to avoid over-/underflow in the eigenvalue equation which defines the first eigenvalue. If the eigenvalues are complex, then the eigenvalues are (WR1 +/- WI i) / SCALE1 (which may lie outside the exponent range of the machine), SCALE1=SCALE2, and SCALE1 will always be positive. If the eigenvalues are real, then the first (real) eigenvalue is WR1 / SCALE1 , but this may overflow or underflow, and in fact, SCALE1 may be zero or less than the underflow threshold if the exact eigenvalue is sufficiently large.

SCALE2 Output parameter.

SCALE2 is DOUBLE PRECISION

A scaling factor used to avoid over-/underflow in the eigenvalue equation which defines the second eigenvalue. If the eigenvalues are complex, then SCALE2=SCALE1. If the eigenvalues are real, then the second (real) eigenvalue is WR2 / SCALE2 , but this may overflow or underflow, and in fact, SCALE2 may be zero or less than the underflow threshold if the exact eigenvalue is sufficiently large.

WR1 Output parameter.

WR1 is DOUBLE PRECISION

If the eigenvalue is real, then WR1 is SCALE1 times the eigenvalue closest to the (2,2) element of A B**(-1). If the eigenvalue is complex, then WR1=WR2 is SCALE1 times the real part of the eigenvalues.

WR2 Output parameter.

WR2 is DOUBLE PRECISION

If the eigenvalue is real, then WR2 is SCALE2 times the other eigenvalue. If the eigenvalue is complex, then WR1=WR2 is SCALE1 times the real part of the eigenvalues.

WI Output parameter.

WI is DOUBLE PRECISION

If the eigenvalue is real, then WI is zero. If the eigenvalue is complex, then WI is SCALE1 times the imaginary part of the eigenvalues. WI will always be non-negative.

Related Information

For this routine in other precisions, please see *slag2*.

4.17.245 dlag2s

dlag2s converts a DOUBLE PRECISION matrix, SA, to a SINGLE PRECISION matrix, A.

RMAX is the overflow for the SINGLE PRECISION arithmetic dlag2s checks that all the entries of A are between -RMAX and RMAX. If not the conversion is aborted and a flag is raised.

This is an auxiliary routine so there is no argument checking.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlag2s(M, N, A, LDA, SA, LDSA, INFO)
```

C specification:

```
#include "armpl.h"

void dlag2s_(const armpl_int_t *m, const armpl_int_t *n, const double *a,
             const armpl_int_t *lda, float *sa, const armpl_int_t *ldsa,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of lines of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the M-by-N coefficient matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

SA Output parameter.

SA is REAL

SA is an array, dimension (LDSA, N). On exit, if INFO=0, the M-by-N coefficient matrix SA; if INFO>0, the content of SA is unspecified.

LDSA Input parameter.

LDSA is INTEGER

The leading dimension of the array SA. LDSA >= max(1, M).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. = 1: an entry of the matrix A is greater than the SINGLE PRECISION overflow threshold, in this case, the content of SA in exit is unspecified.

Related Information

It also exists with a native C interface as *LAPACKE_dlag2s*.

4.17.246 dlags2

dlags2 computes 2-by-2 orthogonal matrices U, V and Q, such that if (UPPER) then

```
U**T *A*Q = U**T *( A1 A2 )*Q = ( x  0  )
                  ( 0  A3 )     ( x  x  )
```

and

```
V**T*B*Q = V**T *( B1 B2 )*Q = ( x  0  )
                 ( 0  B3 )     ( x  x  )
```

or if (.NOT.UPPER) then

```
U**T *A*Q = U**T *( A1 0  )*Q = ( x  x  )
                  ( A2 A3 )     ( 0  x  )
```

and

```
V**T*B*Q = V**T*( B1 0  )*Q = ( x  x  )
                ( B2 B3 )     ( 0  x  )
```

The rows of the transformed A and B are parallel, where

```
U = (  CSU  SNU ), V = (  CSV SNV ), Q = (  CSQ   SNQ )
    ( -SNU  CSU )      ( -SNV CSV )      ( -SNQ   CSQ )
```

Z^T denotes the transpose of Z.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlags2(UPPER, A1, A2, A3, B1, B2, B3, CSU, SNU, CSV, SNV, CSQ,
                  SNQ)
```

C specification:

```
#include "armpl.h"

void dlags2_(const armpl_int_t *upper, const double *a1, const double *a2,
             const double *a3, const double *b1, const double *b2,
             const double *b3, double *csu, double *snu, double *csv,
             double *snv, double *csq, double *snq);
```

Parameters

UPPER Input parameter.

UPPER is LOGICAL

= .TRUE.: the input matrices A and B are upper triangular. = .FALSE.: the input matrices A and B are lower triangular.

A1 Input parameter.

A1 is DOUBLE PRECISION

A2 Input parameter.

A2 is DOUBLE PRECISION

A3 Input parameter.

A3 is DOUBLE PRECISION

On entry, A1, A2 and A3 are elements of the input 2-by-2 upper (lower) triangular matrix A.

B1 Input parameter.

B1 is DOUBLE PRECISION

B2 Input parameter.

B2 is DOUBLE PRECISION

B3 Input parameter.

B3 is DOUBLE PRECISION

On entry, B1, B2 and B3 are elements of the input 2-by-2 upper (lower) triangular matrix B.

CSU Output parameter.

CSU is DOUBLE PRECISION

SNU Output parameter.

SNU is DOUBLE PRECISION

The desired orthogonal matrix U.

CSV Output parameter.

CSV is DOUBLE PRECISION

SNV Output parameter.

SNV is DOUBLE PRECISION

The desired orthogonal matrix V.

CSQ Output parameter.

CSQ is DOUBLE PRECISION

SNQ Output parameter.

SNQ is DOUBLE PRECISION

The desired orthogonal matrix Q.

Related Information

For this routine in other precisions, please see *clags2*, *slags2* and *zlags2*.

4.17.247 dlagtf

dlagtf factorizes the matrix (T - lambda*I), where T is an n by n tridiagonal matrix and lambda is a scalar, as

```
T - lambda*I = PLU,
```

where P is a permutation matrix, L is a unit lower tridiagonal matrix with at most one non-zero sub-diagonal elements per column and U is an upper triangular matrix with at most two non-zero super-diagonal elements per column.

The factorization is obtained by Gaussian elimination with partial pivoting and implicit row scaling.

The parameter LAMBDA is included in the routine so that dlagtf may be used, in conjunction with DLAGTS, to obtain eigenvectors of T by inverse iteration.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlagtf(N, A, LAMBDA, B, C, TOL, D, IN, INFO)
```

C specification:

```
#include "armpl.h"

void dlagtf_(const armpl_int_t *n, double *a, const double *lambda, double *b,
             double *c, const double *tol, double *d, armpl_int_t *in,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix T.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (N). On entry, A must contain the diagonal elements of T.

On exit, A is overwritten by the n diagonal elements of the upper triangular matrix U of the factorization of T.

LAMBDA Input parameter.

LAMBDA is DOUBLE PRECISION

On entry, the scalar lambda.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (N-1). On entry, B must contain the (n-1) super-diagonal elements of T.

On exit, B is overwritten by the (n-1) super-diagonal elements of the matrix U of the factorization of T.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (N-1). On entry, C must contain the (n-1) sub-diagonal elements of T.

On exit, C is overwritten by the (n-1) sub-diagonal elements of the matrix L of the factorization of T.

TOL Input parameter.

TOL is DOUBLE PRECISION

On entry, a relative tolerance used to indicate whether or not the matrix (T - lambda*I) is nearly singular. TOL should normally be chose as approximately the largest relative error in the elements of T. For example, if the elements of T are correct to about 4 significant figures, then TOL should be set to about 5*10**(-4). If TOL is supplied as less than eps, where eps is the relative machine precision, then the value eps is used in place of TOL.

D Output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N-2). On exit, D is overwritten by the (n-2) second super-diagonal elements of the matrix U of the factorization of T.

IN Output parameter.

IN is INTEGER array, dimension (N)

On exit, IN contains details of the permutation matrix P. If an interchange occurred at the kth step of the elimination, then IN(k) = 1, otherwise IN(k) = 0. The element IN(n) returns the smallest positive integer j such that

abs(u(j,j)).le. norm((T - lambda*I)(j))*TOL,

where norm(A(j)) denotes the sum of the absolute values of the jth row of the matrix A. If no such j exists then IN(n) is returned as zero. If IN(n) is returned as positive, then a diagonal element of U is small, indicating that (T - lambda*I) is singular or nearly singular,

INFO Output parameter.

INFO is INTEGER

= 0 : successful exit .lt. 0: if INFO = -k, the kth argument had an illegal value

Related Information

For this routine in other precisions, please see *slagtf*.

4.17.248 dlagtm

dlagtm performs a matrix-vector product of the form

```
B := alpha * A * X + beta * B
```

where A is a tridiagonal matrix of order N, B and X are N by NRHS matrices, and alpha and beta are real scalars, each of which may be 0., 1., or -1.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlagtm(TRANS, N, NRHS, ALPHA, DL, D, DU, X, LDX, BETA, B, LDB)
```

C specification:

```
#include "armpl.h"

void dlagtm_(const char *trans, const armpl_int_t *n, const armpl_int_t *nrhs,
             const double *alpha, const double *dl, const double *d,
             const double *du, const double *x, const armpl_int_t *ldx,
             const double *beta, double *b, const armpl_int_t *ldb, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the operation applied to A. = 'N': No transpose, B := alpha * A * X + beta * B = 'T': Transpose, B := alpha * A'* X + beta * B = 'C': Conjugate transpose = Transpose

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices X and B.

ALPHA Input parameter.

ALPHA is DOUBLE PRECISION

The scalar alpha. ALPHA must be 0., 1., or -1.; otherwise, it is assumed to be 0.

DL Input parameter.

DL is DOUBLE PRECISION

DL is an array, dimension (N-1). The (n-1) sub-diagonal elements of T.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The diagonal elements of T.

DU Input parameter.

DU is DOUBLE PRECISION

DU is an array, dimension (N-1). The (n-1) super-diagonal elements of T.

X Input parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, NRHS). The N by NRHS matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(N,1).

BETA Input parameter.

BETA is DOUBLE PRECISION

The scalar beta. BETA must be 0., 1., or -1.; otherwise, it is assumed to be 1.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the N by NRHS matrix B. On exit, B is overwritten by the matrix expression B := alpha * A * X + beta * B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(N,1).

Related Information

For this routine in other precisions, please see *clagtm*, *slagtm* and *zlagtm*.

4.17.249 dlagts

`dlagts` may be used to solve one of the systems of equations

```
(T - lambda*I)*x = y   or   (T - lambda*I)**T*x = y,
```

where T is an n by n tridiagonal matrix, for x, following the factorization of (T - lambda*I) as

```
(T - lambda*I) = P*L*U ,
```

by routine DLAGTF. The choice of equation to be solved is controlled by the argument JOB, and in each case there is an option to perturb zero or very small diagonal elements of U, this option being intended for use in applications such as inverse iteration.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlagts(JOB, N, A, B, C, D, IN, Y, TOL, INFO)
```

C specification:

```
#include "armpl.h"

void dlagts_(const armpl_int_t *job, const armpl_int_t *n, const double *a,
             const double *b, const double *c, const double *d,
             const armpl_int_t *in, double *y, double *tol,
             armpl_int_t *info);
```

Parameters

JOB Input parameter.

JOB is INTEGER

Specifies the job to be performed by DLAGTS as follows: = 1: The equations (T - lambda*I)x = y are to be solved, but diagonal elements of U are not to be perturbed. = -1: The equations (T - lambda*I)x = y are to be solved and, if overflow would otherwise occur, the diagonal elements of U are to be perturbed. See argument TOL below. = 2: The equations $(T - lambda*I)^T x = y$ are to be solved, but diagonal elements of U are not to be perturbed. = -2: The equations $(T - lambda*I)^T x = y$ are to be solved and, if overflow would otherwise occur, the diagonal elements of U are to be perturbed. See argument TOL below.

N Input parameter.

N is INTEGER

The order of the matrix T.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (N). On entry, A must contain the diagonal elements of U as returned from DLAGTF.

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (N-1). On entry, B must contain the first super-diagonal elements of U as returned from DLAGTF.

C Input parameter.

C is DOUBLE PRECISION

C is an array, dimension (N-1). On entry, C must contain the sub-diagonal elements of L as returned from DLAGTF.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (N-2). On entry, D must contain the second super-diagonal elements of U as returned from DLAGTF.

IN Input parameter.

IN is INTEGER array, dimension (N)

On entry, IN must contain details of the matrix P as returned from DLAGTF.

Y Input and output parameter.

Y is DOUBLE PRECISION

Y is an array, dimension (N). On entry, the right hand side vector y. On exit, Y is overwritten by the solution vector x.

TOL Input and output parameter.

TOL is DOUBLE PRECISION

On entry, with JOB .lt. 0, TOL should be the minimum perturbation to be made to very small diagonal elements of U. TOL should normally be chosen as about eps*norm(U), where eps is the relative machine precision, but if TOL is supplied as non-positive, then it is reset to eps*max(abs(u(i,j))). If JOB .gt. 0 then TOL is not referenced.

On exit, TOL is changed as described above, only if TOL is non-positive on entry. Otherwise TOL is unchanged.

INFO Output parameter.

INFO is INTEGER

= 0 : successful exit .lt. 0: if INFO = -i, the i-th argument had an illegal value .gt. 0: overflow would occur when computing the INFO(th) element of the solution vector x. This can only occur when JOB is supplied as positive and either means that a diagonal element of U is very small, or that the elements of the right-hand side vector y are very large.

Related Information

For this routine in other precisions, please see *slagts*.

4.17.250 dlagv2

dlagv2 computes the Generalized Schur factorization of a real 2-by-2 matrix pencil (A,B) where B is upper triangular. This routine computes orthogonal (rotation) matrices given by CSL, SNL and CSR, SNR such that

1) if the pencil (A,B) has two real eigenvalues (include 0/0 or 1/0

```
 types), then

[ a11 a12 ] := [  CSL  SNL ] [ a11 a12 ] [  CSR -SNR ]
[  0  a22 ]    [ -SNL  CSL ] [ a21 a22 ] [  SNR  CSR ]

[ b11 b12 ] := [  CSL  SNL ] [ b11 b12 ] [  CSR -SNR ]
[  0  b22 ]    [ -SNL  CSL ] [  0  b22 ] [  SNR  CSR ],
```

2) if the pencil (A,B) has a pair of complex conjugate eigenvalues,

```
 then

[ a11 a12 ] := [  CSL  SNL ] [ a11 a12 ] [  CSR -SNR ]
[ a21 a22 ]    [ -SNL  CSL ] [ a21 a22 ] [  SNR  CSR ]

[ b11  0  ] := [  CSL  SNL ] [ b11 b12 ] [  CSR -SNR ]
[  0  b22 ]    [ -SNL  CSL ] [  0  b22 ] [  SNR  CSR ]

where b11 >= b22 > 0.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dlagv2(A, LDA, B, LDB, ALPHAR, ALPHAI, BETA, CSL, SNL, CSR, SNR)
```

C specification:

```
#include "armpl.h"

void dlagv2_(double *a, const armpl_int_t *lda, double *b,
             const armpl_int_t *ldb, double *alphar, double *alphai,
             double *beta, double *csl, double *snl, double *csr,
             double *snr);
```

Parameters

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, 2). On entry, the 2 x 2 matrix A. On exit, A is overwritten by the ''A-part" of the generalized Schur form.

LDA Input parameter.

LDA is INTEGER

THe leading dimension of the array A. LDA >= 2.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, 2). On entry, the upper triangular 2 x 2 matrix B. On exit, B is overwritten by the ''B-part" of the generalized Schur form.

LDB Input parameter.

LDB is INTEGER

THe leading dimension of the array B. LDB >= 2.

ALPHAR Output parameter.

ALPHAR is DOUBLE PRECISION

ALPHAR is an array, dimension (2) .

ALPHAI Output parameter.

ALPHAI is DOUBLE PRECISION

ALPHAI is an array, dimension (2) .

BETA Output parameter.

BETA is DOUBLE PRECISION

BETA is an array, dimension (2). (ALPHAR(k)+i*ALPHAI(k))/BETA(k) are the eigenvalues of the pencil (A, B), k=1,2, i = sqrt(-1). Note that BETA(k) may be zero.

CSL Output parameter.

CSL is DOUBLE PRECISION

The cosine of the left rotation matrix.

SNL Output parameter.

SNL is DOUBLE PRECISION

The sine of the left rotation matrix.

CSR Output parameter.

CSR is DOUBLE PRECISION

The cosine of the right rotation matrix.

SNR Output parameter.

SNR is DOUBLE PRECISION

The sine of the right rotation matrix.

Related Information

For this routine in other precisions, please see *slagv2*.

4.17.251 dlahqr

```
DLAHQR is an auxiliary routine called by DHSEQR to update the
   eigenvalues and Schur decomposition already computed by DHSEQR, by
   dealing with the Hessenberg submatrix in rows and columns ILO to
   IHI.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dlahqr(WANTT, WANTZ, N, ILO, IHI, H, LDH, WR, WI, ILOZ, IHIZ, Z,
                  LDZ, INFO)
```

C specification:

```
#include "armpl.h"

void dlahqr_(const armpl_int_t *wantt, const armpl_int_t *wantz,
             const armpl_int_t *n, const armpl_int_t *ilo,
             const armpl_int_t *ihi, double *h, const armpl_int_t *ldh,
             double *wr, double *wi, const armpl_int_t *iloz,
             const armpl_int_t *ihiz, double *z, const armpl_int_t *ldz,
             armpl_int_t *info);
```

Parameters

WANTT Input parameter.

WANTT is LOGICAL

= .TRUE. : the full Schur form T is required; = .FALSE.: only eigenvalues are required.

WANTZ Input parameter.

WANTZ is LOGICAL

= .TRUE. : the matrix of Schur vectors Z is required; = .FALSE.: Schur vectors are not required.

N Input parameter.

N is INTEGER

The order of the matrix H. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

It is assumed that H is already upper quasi-triangular in rows and columns IHI+1:N, and that H(ILO,ILO-1) = 0 (unless ILO = 1). DLAHQR works primarily with the Hessenberg submatrix in rows and columns ILO to IHI, but applies transformations to all of H if WANTT is .TRUE.. 1 <= ILO <= max(1, IHI); IHI <= N.

H Input and output parameter.

H is DOUBLE PRECISION

H is an array, dimension (LDH, N). On entry, the upper Hessenberg matrix H. On exit, if INFO is zero and if WANTT is .TRUE., H is upper quasi-triangular in rows and columns ILO:IHI, with any 2-by-2 diagonal blocks in standard form. If INFO is zero and WANTT is .FALSE., the contents of H are unspecified on exit. The output state of H if INFO is nonzero is given below under the description of INFO.

LDH Input parameter.

LDH is INTEGER

The leading dimension of the array H. LDH >= max(1, N).

WR Output parameter.

WR is DOUBLE PRECISION

WR is an array, dimension (N) .

WI Output parameter.

WI is DOUBLE PRECISION

WI is an array, dimension (N). The real and imaginary parts, respectively, of the computed eigenvalues ILO to IHI are stored in the corresponding elements of WR and WI. If two eigenvalues are computed as a complex conjugate pair, they are stored in consecutive elements of WR and WI, say the i-th and (i+1)th, with WI(i) > 0 and WI(i+1) < 0. If WANTT is .TRUE., the eigenvalues are stored in the same order as on the diagonal of the Schur form returned in H, with WR(i) = H(i,i), and, if H(i:i+1,i:i+1) is a 2-by-2 diagonal block, WI(i) = sqrt(H(i+1,i)*H(i,i+1)) and WI(i+1) = -WI(i).

ILOZ Input parameter.

ILOZ is INTEGER

IHIZ Input parameter.

IHIZ is INTEGER

Specify the rows of Z to which transformations must be applied if WANTZ is .TRUE.. 1 <= ILOZ <= ILO; IHI <= IHIZ <= N.

Z Input and output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, N). If WANTZ is .TRUE., on entry Z must contain the current matrix Z of transformations accumulated by DHSEQR, and on exit Z has been updated; transformations are applied only to the submatrix Z(ILOZ:IHIZ,ILO:IHI). If WANTZ is .FALSE., Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit .GT. 0: If INFO = i, DLAHQR failed to compute all the eigenvalues ILO to IHI in a total of 30 iterations per eigenvalue; elements i+1:ihi of WR and WI contain those eigenvalues which have been successfully computed.

If INFO .GT. 0 and WANTT is .FALSE., then on exit, the remaining unconverged eigenvalues are the eigenvalues of the upper Hessenberg matrix rows and columns ILO thorugh INFO of the final, output value of H.

If INFO .GT. 0 and WANTT is .TRUE., then on exit (*) (initial value of H)*U = U*(final value of H) where U is an orthognal matrix. The final value of H is upper Hessenberg and triangular in rows and columns INFO+1 through IHI.

If INFO .GT. 0 and WANTZ is .TRUE., then on exit (final value of Z) = (initial value of Z)*U where U is the orthogonal matrix in (*) (regardless of the value of WANTT.)

Related Information

For this routine in other precisions, please see *clahqr*, *slahqr* and *zlahqr*.

4.17.252 dlahr2

dlahr2 reduces the first NB columns of A real general n-BY-(n-k+1) matrix A so that elements below the k-th subdiagonal are zero. The reduction is performed by an orthogonal similarity transformation Q^T * A * Q. The routine returns the matrices V and T which determine Q as a block reflector I - V*T*V^T , and also the matrix Y = A * V * T.

This is an auxiliary routine called by DGEHRD.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlahr2(N, K, NB, A, LDA, TAU, T, LDT, Y, LDY)
```

C specification:

```
#include "armpl.h"

void dlahr2_(const armpl_int_t *n, const armpl_int_t *k,
```

(continues on next page)

(continued from previous page)

```
                const armpl_int_t *nb, double *a, const armpl_int_t *lda,
                double *tau, double *t, const armpl_int_t *ldt, double *y,
                const armpl_int_t *ldy);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A.

K Input parameter.

K is INTEGER

The offset for the reduction. Elements below the k-th subdiagonal in the first NB columns are reduced to zero. K < N.

NB Input parameter.

NB is INTEGER

The number of columns to be reduced.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA,N-K+1). On entry, the n-by-(n-k+1) general matrix A. On exit, the elements on and above the k-th subdiagonal in the first NB columns are overwritten with the corresponding elements of the reduced matrix; the elements below the k-th subdiagonal, with the array TAU, represent the matrix Q as a product of elementary reflectors. The other columns of A are unchanged. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TAU Output parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (NB). The scalar factors of the elementary reflectors. See Further Details.

T Output parameter.

T is DOUBLE PRECISION

T is an array, dimension (LDT, NB). The upper triangular matrix T.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= NB.

Y Output parameter.

Y is DOUBLE PRECISION

Y is an array, dimension (LDY, NB). The n-by-nb matrix Y.

LDY Input parameter.

LDY is INTEGER

The leading dimension of the array Y. LDY >= N.

Related Information

For this routine in other precisions, please see *clahr2*, *slahr2* and *zlahr2*.

4.17.253 dlaic1

`dlaic1` applies one step of incremental condition estimation in its simplest version:

Let x, twonorm(x) = 1, be an approximate singular vector of an j-by-j lower triangular matrix L, such that

```
twonorm(L*x) = sest
```

Then `dlaic1` computes sestpr, s, c such that the vector

```
       [ s*x ]
xhat = [  c  ]
```

is an approximate singular vector of

```
       [ L       0  ]
Lhat = [ w**T gamma ]
```

in the sense that

```
twonorm(Lhat*xhat) = sestpr.
```

Depending on JOB, an estimate for the largest or smallest singular value is computed.

Note that [s c]^T and sestpr**2 is an eigenpair of the system

```
diag(sest*sest, 0) + [alpha  gamma] * [ alpha ]
                                      [ gamma ]
```

where alpha = x^T *w.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlaic1(JOB, J, X, SEST, W, GAMMA, SESTPR, S, C)
```

C specification:

```
#include "armpl.h"

void dlaic1_(const armpl_int_t *job, const armpl_int_t *j, const double *x,
             const double *sest, const double *w, const double *gamma,
             double *sestpr, double *s, double *c);
```

Parameters

JOB Input parameter.

JOB is INTEGER

= 1: an estimate for the largest singular value is computed. = 2: an estimate for the smallest singular value is computed.

J Input parameter.

J is INTEGER

Length of X and W

X Input parameter.

X is DOUBLE PRECISION

X is an array, dimension (J). The j-vector x.

SEST Input parameter.

SEST is DOUBLE PRECISION

Estimated singular value of j by j matrix L

W Input parameter.

W is DOUBLE PRECISION

W is an array, dimension (J). The j-vector w.

GAMMA Input parameter.

GAMMA is DOUBLE PRECISION

The diagonal element gamma.

SESTPR Output parameter.

SESTPR is DOUBLE PRECISION

Estimated singular value of (j+1) by (j+1) matrix Lhat.

S Output parameter.

S is DOUBLE PRECISION

Sine needed in forming xhat.

C Output parameter.

C is DOUBLE PRECISION

Cosine needed in forming xhat.

Related Information

For this routine in other precisions, please see *claic1*, *slaic1* and *zlaic1*.

4.17.254 dlaisnan

This routine is not for general use. It exists solely to avoid over-optimization in DISNAN.

DLAISNAN checks for NaNs by comparing its two arguments for inequality. NaN is the only floating-point value where NaN != NaN returns .TRUE. To check for NaNs, pass the same variable as both arguments.

A compiler must assume that the two arguments are not the same variable, and the test will not be optimized away. Interprocedural or whole-program optimization may delete this test. The ISNAN functions will be replaced by the correct Fortran 03 intrinsic once the intrinsic is widely available.

Syntax

Fortran specification:

```
use armpl_library

logical function dlaisnan(DIN1, DIN2)
```

C specification:

```
#include "armpl.h"

armpl_int_t dlaisnan_(const double *din1, const double *din2);
```

Parameters

DIN1 Input parameter.

DIN1 is DOUBLE PRECISION

DIN2 Input parameter.

DIN2 is DOUBLE PRECISION

Two numbers to compare for inequality.

Related Information

For this routine in other precisions, please see *slaisnan*.

4.17.255 dlaln2

dlaln2 solves a system of the form (ca A - w D) X = s B or (ca A^T - w D) X = s B with possible scaling ("s") and perturbation of A. (A^T means A-transpose.)

A is an NA x NA real matrix, ca is a real scalar, D is an NA x NA real diagonal matrix, w is a real or complex value, and X and B are NA x 1 matrices – real if w is real, complex if w is complex. NA may be 1 or 2.

If w is complex, X and B are represented as NA x 2 matrices, the first column of each being the real part and the second being the imaginary part.

"s" is a scaling factor (.LE. 1), computed by dlaln2, which is so chosen that X can be computed without overflow. X is further scaled if necessary to assure that norm(ca A - w D)*norm(X) is less than overflow.

If both singular values of (ca A - w D) are less than SMIN, SMIN*identity will be used instead of (ca A - w D). If only one singular value is less than SMIN, one element of (ca A - w D) will be perturbed enough to make the smallest singular value roughly SMIN. If both singular values are at least SMIN, (ca A - w D) will not be perturbed. In any case, the perturbation will be at most some small multiple of max(SMIN, ulp*norm(ca A - w D)). The singular values are computed by infinity-norm approximations, and thus will only be correct to a factor of 2 or so.

Note: all input quantities are assumed to be smaller than overflow by a reasonable factor. (See BIGNUM.)

Syntax

Fortran specification:

```
use armpl_library

subroutine dlaln2(LTRANS, NA, NW, SMIN, CA, A, LDA, D1, D2, B, LDB, WR, WI, X,
                  LDX, SCALE, XNORM, INFO)
```

C specification:

```
#include "armpl.h"

void dlaln2_(const armpl_int_t *ltrans, const armpl_int_t *na,
             const armpl_int_t *nw, const double *smin, const double *ca,
             const double *a, const armpl_int_t *lda, const double *d1,
             const double *d2, const double *b, const armpl_int_t *ldb,
             const double *wr, const double *wi, double *x,
             const armpl_int_t *ldx, double *scale, double *xnorm,
             armpl_int_t *info);
```

Parameters

LTRANS Input parameter.

LTRANS is LOGICAL

=.TRUE.: A-transpose will be used. =.FALSE.: A will be used (not transposed.)

NA Input parameter.

NA is INTEGER

The size of the matrix A. It may (only) be 1 or 2.

NW Input parameter.

NW is INTEGER

1 if “w” is real, 2 if “w” is complex. It may only be 1 or 2.

SMIN Input parameter.

SMIN is DOUBLE PRECISION

The desired lower bound on the singular values of A. This should be a safe distance away from underflow or overflow, say, between (underflow/machine precision) and (machine precision * overflow). (See BIGNUM and ULP.)

CA Input parameter.

CA is DOUBLE PRECISION

The coefficient c, which A is multiplied by.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, NA). The NA x NA matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of A. It must be at least NA.

D1 Input parameter.

D1 is DOUBLE PRECISION

The 1,1 element in the diagonal matrix D.

D2 Input parameter.

D2 is DOUBLE PRECISION

The 2,2 element in the diagonal matrix D. Not used if NA=1.

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NW). The NA x NW matrix B (right-hand side). If NW=2 (“w” is complex), column 1 contains the real part of B and column 2 contains the imaginary part.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B. It must be at least NA.

WR Input parameter.

WR is DOUBLE PRECISION

The real part of the scalar “w”.

WI Input parameter.

WI is DOUBLE PRECISION

The imaginary part of the scalar “w”. Not used if NW=1.

X Output parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, NW). The NA x NW matrix X (unknowns), as computed by DLALN2. If NW=2 (“w” is complex), on exit, column 1 will contain the real part of X and column 2 will contain the imaginary part.

LDX Input parameter.

LDX is INTEGER

The leading dimension of X. It must be at least NA.

SCALE Output parameter.

SCALE is DOUBLE PRECISION

The scale factor that B must be multiplied by to insure that overflow does not occur when computing X. Thus, (ca A - w D) X will be SCALE*B, not B (ignoring perturbations of A.) It will be at most 1.

XNORM Output parameter.

XNORM is DOUBLE PRECISION

The infinity-norm of X, when X is regarded as an NA x NW real matrix.

INFO Output parameter.

INFO is INTEGER

An error flag. It will be set to zero if no error occurs, a negative number if an argument is in error, or a positive number if ca A - w D had to be perturbed. The possible values are: = 0: No error occurred, and (ca A - w D) did not have to be perturbed. = 1: (ca A - w D) had to be perturbed to make its smallest (or only) singular value greater than SMIN. NOTE: In the interests of speed, this routine does not check the inputs for errors.

Related Information

For this routine in other precisions, please see *slaln2*.

4.17.256 dlals0

`dlals0` applies back the multiplying factors of either the left or the right singular vector matrix of a diagonal matrix appended by a row to the right hand side matrix B in solving the least squares problem using the divide-and-conquer SVD approach.

For the left singular vector matrix, three types of orthogonal matrices are involved:

(1L) Givens rotations: the number of such rotations is GIVPTR; the

```
pairs of columns/rows they were applied to are stored in GIVCOL;
and the C- and S-values of these rotations are stored in GIVNUM.
```

(2L) Permutation. The (NL+1)-st row of B is to be moved to the first

```
row, and for J=2:N, PERM(J)-th row of B is to be moved to the
J-th row.
```

(3L) The left singular vector matrix of the remaining matrix.

For the right singular vector matrix, four types of orthogonal matrices are involved:

(1R) The right singular vector matrix of the remaining matrix.

(2R) If SQRE = 1, one extra Givens rotation to generate the right

```
null space.
```

(3R) The inverse transformation of (2L).

(4R) The inverse transformation of (1L).

Syntax

Fortran specification:

```
use armpl_library

subroutine dlals0(ICOMPQ, NL, NR, SQRE, NRHS, B, LDB, BX, LDBX, PERM, GIVPTR,
                  GIVCOL, LDGCOL, GIVNUM, LDGNUM, POLES, DIFL, DIFR, Z, K, C,
                  S, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dlals0_(const armpl_int_t *icompq, const armpl_int_t *nl,
             const armpl_int_t *nr, const armpl_int_t *sqre,
             const armpl_int_t *nrhs, double *b, const armpl_int_t *ldb,
             double *bx, const armpl_int_t *ldbx, const armpl_int_t *perm,
             const armpl_int_t *givptr, const armpl_int_t *givcol,
             const armpl_int_t *ldgcol, const double *givnum,
             const armpl_int_t *ldgnum, const double *poles,
             const double *difl, const double *difr, const double *z,
             const armpl_int_t *k, const double *c, const double *s,
             double *work, armpl_int_t *info);
```

Parameters

ICOMPQ Input parameter.

ICOMPQ is INTEGER

Specifies whether singular vectors are to be computed in factored form: = 0: Left singular vector matrix. = 1: Right singular vector matrix.

NL Input parameter.

NL is INTEGER

The row dimension of the upper block. NL >= 1.

NR Input parameter.

NR is INTEGER

The row dimension of the lower block. NR >= 1.

SQRE Input parameter.

SQRE is INTEGER

= 0: the lower block is an NR-by-NR square matrix. = 1: the lower block is an NR-by-(NR+1) rectangular matrix.

The bidiagonal matrix has row dimension N = NL + NR + 1, and column dimension M = N + SQRE.

NRHS Input parameter.

NRHS is INTEGER

The number of columns of B and BX. NRHS must be at least 1.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On input, B contains the right hand sides of the least squares problem in rows 1 through M. On output, B contains the solution X in rows 1 through N.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B. LDB must be at least max(1,MAX(M, N)).

BX Output parameter.

BX is DOUBLE PRECISION

BX is an array, dimension (LDBX, NRHS) .

LDBX Input parameter.

LDBX is INTEGER

The leading dimension of BX.

PERM Input parameter.

PERM is INTEGER array, dimension (N)

The permutations (from deflation and sorting) applied to the two blocks.

GIVPTR Input parameter.

GIVPTR is INTEGER

The number of Givens rotations which took place in this subproblem.

GIVCOL Input parameter.

GIVCOL is INTEGER array, dimension (LDGCOL, 2)

Each pair of numbers indicates a pair of rows/columns involved in a Givens rotation.

LDGCOL Input parameter.

LDGCOL is INTEGER

The leading dimension of GIVCOL, must be at least N.

GIVNUM Input parameter.

GIVNUM is DOUBLE PRECISION

GIVNUM is an array, dimension (LDGNUM, 2). Each number indicates the C or S value used in the corresponding Givens rotation.

LDGNUM Input parameter.

LDGNUM is INTEGER

The leading dimension of arrays DIFR, POLES and GIVNUM, must be at least K.

POLES Input parameter.

POLES is DOUBLE PRECISION

POLES is an array, dimension (LDGNUM, 2). On entry, POLES(1:K, 1) contains the new singular values obtained from solving the secular equation, and POLES(1:K, 2) is an array containing the poles in the secular equation.

DIFL Input parameter.

DIFL is DOUBLE PRECISION

DIFL is an array, dimension (K).. On entry, DIFL(I) is the distance between I-th updated (undeflated) singular value and the I-th (undeflated) old singular value.

DIFR Input parameter.

DIFR is DOUBLE PRECISION

DIFR is an array, dimension (LDGNUM, 2).. On entry, DIFR(I, 1) contains the distances between I-th updated (undeflated) singular value and the I+1-th (undeflated) old singular value. And DIFR(I, 2) is the normalizing factor for the I-th right singular vector.

Z Input parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (K). Contain the components of the deflation-adjusted updating row vector.

K Input parameter.

K is INTEGER

Contains the dimension of the non-deflated matrix, This is the order of the related secular equation. 1 <= K <=N.

C Input parameter.

C is DOUBLE PRECISION

C contains garbage if SQRE =0 and the C-value of a Givens rotation related to the right null space if SQRE = 1.

S Input parameter.

S is DOUBLE PRECISION

S contains garbage if SQRE =0 and the S-value of a Givens rotation related to the right null space if SQRE = 1.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (K) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *clals0*, *slals0* and *zlals0*.

4.17.257 dlalsa

dlalsa is an itermediate step in solving the least squares problem by computing the SVD of the coefficient matrix in compact form (The singular vectors are computed as products of simple orthorgonal matrices.).

If ICOMPQ = 0, dlalsa applies the inverse of the left singular vector matrix of an upper bidiagonal matrix to the right hand side; and if ICOMPQ = 1, dlalsa applies the right singular vector matrix to the right hand side. The singular vector matrices were generated in compact form by dlalsa.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlalsa(ICOMPQ, SMLSIZ, N, NRHS, B, LDB, BX, LDBX, U, LDU, VT, K,
                  DIFL, DIFR, Z, POLES, GIVPTR, GIVCOL, LDGCOL, PERM, GIVNUM,
                  C, S, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dlalsa_(const armpl_int_t *icompq, const armpl_int_t *smlsiz,
             const armpl_int_t *n, const armpl_int_t *nrhs, double *b,
             const armpl_int_t *ldb, double *bx, const armpl_int_t *ldbx,
             const double *u, const armpl_int_t *ldu, const double *vt,
             const armpl_int_t *k, const double *difl, const double *difr,
             const double *z, const double *poles, const armpl_int_t *givptr,
             const armpl_int_t *givcol, const armpl_int_t *ldgcol,
             const armpl_int_t *perm, const double *givnum, const double *c,
             const double *s, double *work, armpl_int_t *iwork,
             armpl_int_t *info);
```

Parameters

ICOMPQ Input parameter.

ICOMPQ is INTEGER

Specifies whether the left or the right singular vector matrix is involved. = 0: Left singular vector matrix = 1: Right singular vector matrix

SMLSIZ Input parameter.

SMLSIZ is INTEGER

The maximum size of the subproblems at the bottom of the computation tree.

N Input parameter.

N is INTEGER

The row and column dimensions of the upper bidiagonal matrix.

NRHS Input parameter.

NRHS is INTEGER

The number of columns of B and BX. NRHS must be at least 1.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On input, B contains the right hand sides of the least squares problem in rows 1 through M. On output, B contains the solution X in rows 1 through N.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B in the calling subprogram. LDB must be at least max(1,MAX(M, N)).

BX Output parameter.

BX is DOUBLE PRECISION

BX is an array, dimension (LDBX, NRHS). On exit, the result of applying the left or right singular vector matrix to B.

LDBX Input parameter.

LDBX is INTEGER

The leading dimension of BX.

U Input parameter.

U is DOUBLE PRECISION

U is an array, dimension (LDU, SMLSIZ).. On entry, U contains the left singular vector matrices of all subproblems at the bottom level.

LDU Input parameter.

LDU is INTEGER, LDU = > N.

The leading dimension of arrays U, VT, DIFL, DIFR, POLES, GIVNUM, and Z.

VT Input parameter.

VT is DOUBLE PRECISION

VT is an array, dimension (LDU, SMLSIZ+1).. On entry, VT^T contains the right singular vector matrices of all subproblems at the bottom level.

K Input parameter.

K is INTEGER array, dimension (N).

DIFL Input parameter.

DIFL is DOUBLE PRECISION

DIFL is an array, dimension (LDU, NLVL).. where NLVL = INT(log_2 (N/(SMLSIZ+1))) + 1.

DIFR Input parameter.

DIFR is DOUBLE PRECISION

DIFR is an array, dimension (LDU, 2 * NLVL).. On entry, DIFL(*, I) and DIFR(*, 2 * I -1) record distances between singular values on the I-th level and singular values on the (I -1)-th level, and DIFR(*, 2 * I) record the normalizing factors of the right singular vectors matrices of subproblems on I-th level.

Z Input parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDU, NLVL).. On entry, Z(1, I) contains the components of the deflation- adjusted updating row vector for subproblems on the I-th level.

POLES Input parameter.

POLES is DOUBLE PRECISION

POLES is an array, dimension (LDU, 2 * NLVL).. On entry, POLES(*, 2 * I -1: 2 * I) contains the new and old singular values involved in the secular equations on the I-th level.

GIVPTR Input parameter.

GIVPTR is INTEGER array, dimension (N).

On entry, GIVPTR(I) records the number of Givens rotations performed on the I-th problem on the computation tree.

GIVCOL Input parameter.

GIVCOL is INTEGER array, dimension (LDGCOL, 2 * NLVL).

On entry, for each I, GIVCOL(*, 2 * I - 1: 2 * I) records the locations of Givens rotations performed on the I-th level on the computation tree.

LDGCOL Input parameter.

LDGCOL is INTEGER, LDGCOL = > N.

The leading dimension of arrays GIVCOL and PERM.

PERM Input parameter.

PERM is INTEGER array, dimension (LDGCOL, NLVL).

On entry, PERM(*, I) records permutations done on the I-th level of the computation tree.

GIVNUM Input parameter.

GIVNUM is DOUBLE PRECISION

GIVNUM is an array, dimension (LDU, 2 * NLVL).. On entry, GIVNUM(*, 2 *I -1 : 2 * I) records the C- and S- values of Givens rotations performed on the I-th level on the computation tree.

C Input parameter.

C is DOUBLE PRECISION

C is an array, dimension (N).. On entry, if the I-th subproblem is not square, C(I) contains the C-value of a Givens rotation related to the right null space of the I-th subproblem.

S Input parameter.

S is DOUBLE PRECISION

S is an array, dimension (N).. On entry, if the I-th subproblem is not square, S(I) contains the S-value of a Givens rotation related to the right null space of the I-th subproblem.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (3*N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *clalsa*, *slalsa* and *zlalsa*.

4.17.258 dlalsd

dlalsd uses the singular value decomposition of A to solve the least squares problem of finding X to minimize the Euclidean norm of each column of A*X-B, where A is N-by-N upper bidiagonal, and X and B are N-by-NRHS. The solution X overwrites B.

The singular values of A smaller than RCOND times the largest singular value are treated as zero in solving the least squares problem; in this case a minimum norm solution is returned. The actual singular values are returned in D in ascending order.

This code makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray XMP, Cray YMP, Cray C 90, or Cray 2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlalsd(UPLO, SMLSIZ, N, NRHS, D, E, B, LDB, RCOND, RANK, WORK,
                  IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dlalsd_(const char *uplo, const armpl_int_t *smlsiz,
             const armpl_int_t *n, const armpl_int_t *nrhs, double *d,
             double *e, double *b, const armpl_int_t *ldb,
             const double *rcond, armpl_int_t *rank, double *work,
             armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': D and E define an upper bidiagonal matrix. = 'L': D and E define a lower bidiagonal matrix.

SMLSIZ Input parameter.

SMLSIZ is INTEGER

The maximum size of the subproblems at the bottom of the computation tree.

N Input parameter.

N is INTEGER

The dimension of the bidiagonal matrix. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of columns of B. NRHS must be at least 1.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry D contains the main diagonal of the bidiagonal matrix. On exit, if INFO = 0, D contains its singular values.

E Input and output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N-1). Contains the super-diagonal entries of the bidiagonal matrix. On exit, E has been destroyed.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On input, B contains the right hand sides of the least squares problem. On output, B contains the solution X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B in the calling subprogram. LDB must be at least max(1, N).

RCOND Input parameter.

RCOND is DOUBLE PRECISION

The singular values of A less than or equal to RCOND times the largest singular value are treated as zero in solving the least squares problem. If RCOND is negative, machine precision is used instead. For example, if diag(S)*X=B were the least squares problem, where diag(S) is a diagonal matrix of singular values, the solution would be X(i) = B(i) / S(i) if S(i) is greater than RCOND*max(S), and X(i) = 0 if S(i) is less than or equal to RCOND*max(S).

RANK Output parameter.

RANK is INTEGER

The number of singular values of A greater than RCOND times the largest singular value.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension at least. (9* N + 2*N* SMLSIZ + 8*N*NLVL + N* NRHS + (SMLSIZ+1)**2), where NLVL = max(0, INT(log_2 (N/(SMLSIZ+1))) + 1).

IWORK Output parameter.

IWORK is INTEGER array, dimension at least

(3*N*NLVL + 11*N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: The algorithm failed to compute a singular value while working on the submatrix lying in rows and columns INFO/(N+1) through MOD(INFO,N+1).

Related Information

For this routine in other precisions, please see *clalsd*, *slalsd* and *zlalsd*.

4.17.259 dlamrg

dlamrg will create a permutation list which will merge the elements of A (which is composed of two independently sorted sets) into a single set which is sorted in ascending order.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlamrg(N1, N2, A, DTRD1, DTRD2, INDEX)
```

C specification:

```
#include "armpl.h"

void dlamrg_(const armpl_int_t *n1, const armpl_int_t *n2, const double *a,
             const armpl_int_t *dtrd1, const armpl_int_t *dtrd2,
             armpl_int_t *index);
```

Parameters

N1 Input parameter.

N1 is INTEGER

N2 Input parameter.

N2 is INTEGER

These arguments contain the respective lengths of the two sorted lists to be merged.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (N1+N2). The first N1 elements of A contain a list of numbers which are sorted in either ascending or descending order. Likewise for the final N2 elements.

DTRD1 Input parameter.

DTRD1 is INTEGER

DTRD2 Input parameter.

DTRD2 is INTEGER

These are the strides to be taken through the array A. Allowable strides are 1 and -1. They indicate whether a subset of A is sorted in ascending (DTRDx = 1) or descending (DTRDx = -1) order.

INDEX Output parameter.

INDEX is INTEGER array, dimension (N1+N2)

On exit this array will contain a permutation such that if B(I) = A(INDEX(I)) for I=1,N1+N2, then B will be sorted in ascending order.

Related Information

For this routine in other precisions, please see *slamrg*.

4.17.260 dlamswlq

DLAMQRTS overwrites the general real M-by-N matrix C with SIDE = 'L' SIDE = 'R' TRANS = 'N': Q * C C * Q TRANS = 'T': Q^T * C C * Q^T where Q is a real orthogonal matrix defined as the product of blocked elementary reflectors computed by short wide LQ factorization (DLASWLQ)

Syntax

Fortran specification:

```
use armpl_library

subroutine dlamswlq(SIDE, TRANS, M, N, K, MB, NB, A, LDA, T, LDT, C, LDC,
                    WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dlamswlq_(const char *side, const char *trans, const armpl_int_t *m,
               const armpl_int_t *n, const armpl_int_t *k,
               const armpl_int_t *mb, const armpl_int_t *nb, double *a,
               const armpl_int_t *lda, const double *t,
               const armpl_int_t *ldt, double *c, const armpl_int_t *ldc,
               double *work, const armpl_int_t *lwork, armpl_int_t *info,
               ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'T': Transpose, apply Q^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >=0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= M.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. M >= K >= 0;

MB Input parameter.

MB is INTEGER

The row block size to be used in the blocked QR. M >= MB >= 1

NB Input parameter.

NB is INTEGER

The column block size to be used in the blocked QR. NB > M.

NB Input parameter.

NB is INTEGER

The block size to be used in the blocked QR. MB > M.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension. (LDA, M) if SIDE = 'L', (LDA, N) if SIDE = 'R' The i-th row must contain the vector which defines the blocked elementary reflector H(i), for i = 1,2,...,k, as returned by DLASWLQ in the first k rows of its array argument A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDA >= max(1, M); if SIDE = 'R', LDA >= max(1, N).

T Input parameter.

T is DOUBLE PRECISION

T is an array, dimension. (M * Number of blocks(CEIL(N-K/NB-K)), The blocked upper triangular block reflectors stored in compact form as a sequence of upper triangular blocks. See below for further details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= MB.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

(workspace) DOUBLE PRECISION

(workspace) is an array, dimension (MAX(1, LWORK)) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, NB) * MB; if SIDE = 'R', LWORK >= max(1, M) * MB. If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal

size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *clamswlq*, *slamswlq* and *zlamswlq*.

4.17.261 dlamtsqr

DLAMTSQR overwrites the general real M-by-N matrix C with SIDE = 'L' SIDE = 'R' TRANS = 'N': Q * C C * Q TRANS = 'T': Q^T * C C * Q^T where Q is a real orthogonal matrix defined as the product of blocked elementary reflectors computed by tall skinny QR factorization (DLATSQR)

Syntax

Fortran specification:

```
use armpl_library

subroutine dlamtsqr(SIDE, TRANS, M, N, K, MB, NB, A, LDA, T, LDT, C, LDC,
                    WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dlamtsqr_(const char *side, const char *trans, const armpl_int_t *m,
               const armpl_int_t *n, const armpl_int_t *k,
               const armpl_int_t *mb, const armpl_int_t *nb, double *a,
               const armpl_int_t *lda, const double *t,
               const armpl_int_t *ldt, double *c, const armpl_int_t *ldc,
               double *work, const armpl_int_t *lwork, armpl_int_t *info,
               ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'T': Transpose, apply Q^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >=0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. M >= N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. N >= K >= 0;

MB Input parameter.

MB is INTEGER

The block size to be used in the blocked QR. MB > N. (must be the same as DLATSQR)

NB Input parameter.

NB is INTEGER

The column block size to be used in the blocked QR. N >= NB >= 1.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, K). The i-th column must contain the vector which defines the blockedelementary reflector H(i), for i = 1,2,. . . ,k, as returned by DLATSQR in the first k columns of its array argument A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDA >= max(1, M); if SIDE = 'R', LDA >= max(1, N).

T Input parameter.

T is DOUBLE PRECISION

T is an array, dimension. (N * Number of blocks(CEIL(M-K/MB-K)), The blocked upper triangular block reflectors stored in compact form as a sequence of upper triangular blocks. See below for further details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= NB.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

(workspace) DOUBLE PRECISION

(workspace) is an array, dimension (MAX(1, LWORK)) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK.

If SIDE = 'L', LWORK >= max(1, N)*NB; if SIDE = 'R', LWORK >= max(1, MB)*NB. If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *clamtsqr*, *slamtsqr* and *zlamtsqr*.

4.17.262 dlaneg

dlaneg computes the Sturm count, the number of negative pivots encountered while factoring tridiagonal T - sigma I = L D L^T. This implementation works directly on the factors without forming the tridiagonal matrix T. The Sturm count is also the number of eigenvalues of T less than sigma.

This routine is called from DLARRB.

The current routine does not use the PIVMIN parameter but rather requires IEEE-754 propagation of Infinities and NaNs. This routine also has no input range restrictions but does require default exception handling such that x/0 produces Inf when x is non-zero, and Inf/Inf produces NaN. For more information, see:

```
Marques, Riedy, and Voemel, "Benefits of IEEE-754 Features in
Modern Symmetric Tridiagonal Eigensolvers," SIAM Journal on
Scientific Computing, v28, n5, 2006.  DOI 10.1137/050641624
(Tech report version in LAWN 172 with the same title.)
```

Syntax

Fortran specification:

```
use armpl_library

integer function dlaneg(N, D, LLD, SIGMA, PIVMIN, R)
```

C specification:

```
#include "armpl.h"

armpl_int_t dlaneg_(const armpl_int_t *n, const double *d, const double *lld,
                    const double *sigma, const double *pivmin,
                    const armpl_int_t *r);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The N diagonal elements of the diagonal matrix D.

LLD Input parameter.

LLD is DOUBLE PRECISION

LLD is an array, dimension (N-1). The (N-1) elements L(i)*L(i)*D(i).

SIGMA Input parameter.

SIGMA is DOUBLE PRECISION

Shift amount in T - sigma I = L D L^T.

PIVMIN Input parameter.

PIVMIN is DOUBLE PRECISION

The minimum pivot in the Sturm sequence. May be used when zero pivots are encountered on non-IEEE-754 architectures.

R Input parameter.

R is INTEGER

The twist index for the twisted factorization that is used for the negcount.

Related Information

For this routine in other precisions, please see *slaneg*.

4.17.263 dlangb

dlangb returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of an n by n band matrix A, with kl sub-diagonals and ku super-diagonals.

Syntax

Fortran specification:

```
use armpl_library

double precision function dlangb(NORM, N, KL, KU, AB, LDAB, WORK)
```

C specification:

```
#include "armpl.h"

double dlangb_(const char *norm, const armpl_int_t *n, const armpl_int_t *kl,
               const armpl_int_t *ku, const double *ab,
               const armpl_int_t *ldab, double *work, ... );
```

Returns

DLANGB = (max(abs(A(i,j))), NORM = ‘M’ or ‘m’

((norm1(A), NORM = ‘1’, ‘O’ or ‘o’ ((normI(A), NORM = ‘I’ or ‘i’ ((normF(A), NORM = ‘F’, ‘f’, ‘E’ or ‘e’

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in DLANGB as described above.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0. When N = 0, DLANGB is set to zero.

KL Input parameter.

KL is INTEGER

The number of sub-diagonals of the matrix A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of super-diagonals of the matrix A. KU >= 0.

AB Input parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). The band matrix A, stored in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(ku+1+i-j,j) = A(i,j) for max(1,j-ku)<=i<=min(n,j+kl).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KL+KU+1.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= N when NORM = 'I'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *clangb*, *slangb* and *zlangb*.

4.17.264 dlange

dlange returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a real matrix A.

Syntax

Fortran specification:

```
use armpl_library

double precision function dlange(NORM, M, N, A, LDA, WORK)
```

C specification:

```
#include "armpl.h"

double dlange_(const char *norm, const armpl_int_t *m, const armpl_int_t *n,
               const double *a, const armpl_int_t *lda, double *work, ... );
```

Returns

DLANGE = (max(abs(A(i,j))), NORM = 'M' or 'm'

((norm1(A), NORM = '1', 'O' or 'o' ((normI(A), NORM = 'I' or 'i' ((normF(A), NORM = 'F', 'f', 'E' or 'e'

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in DLANGE as described above.

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0. When M = 0, DLANGE is set to zero.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0. When N = 0, DLANGE is set to zero.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). The m by n matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(M,1).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= M when NORM = 'I'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *clange*, *slange* and *zlange*. It also exists with a native C interface as *LAPACKE_dlange*.

4.17.265 dlangt

dlangt returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a real tridiagonal matrix A.

Syntax

Fortran specification:

```
use armpl_library

double precision function dlangt(NORM, N, DL, D, DU)
```

C specification:

```
#include "armpl.h"

double dlangt_(const char *norm, const armpl_int_t *n, const double *dl,
               const double *d, const double *du, ... );
```

Returns

DLANGT = (max(abs(A(i,j))), NORM = ‘M’ or ‘m’

((norm1(A), NORM = ‘1’, ‘O’ or ‘o’ ((normI(A), NORM = ‘I’ or ‘i’ ((normF(A), NORM = ‘F’, ‘f’, ‘E’ or ‘e’

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in DLANGT as described above.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0. When N = 0, DLANGT is set to zero.

DL Input parameter.

DL is DOUBLE PRECISION

DL is an array, dimension (N-1). The (n-1) sub-diagonal elements of A.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The diagonal elements of A.

DU Input parameter.

DU is DOUBLE PRECISION

DU is an array, dimension (N-1). The (n-1) super-diagonal elements of A.

Related Information

For this routine in other precisions, please see *clangt*, *slangt* and *zlangt*.

4.17.266 dlanhs

`dlanhs` returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a Hessenberg matrix A.

Syntax

Fortran specification:

```
use armpl_library

double precision function dlanhs(NORM, N, A, LDA, WORK)
```

C specification:

```
#include "armpl.h"

double dlanhs_(const char *norm, const armpl_int_t *n, const double *a,
               const armpl_int_t *lda, double *work, ... );
```

Returns

DLANHS = (max(abs(A(i,j))), NORM = ‘M’ or ‘m’

((norm1(A), NORM = ‘1’, ‘O’ or ‘o’ ((normI(A), NORM = ‘I’ or ‘i’ ((normF(A), NORM = ‘F’, ‘f’, ‘E’ or ‘e’

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

`NORM` is `CHARACTER*1`

Specifies the value to be returned in DLANHS as described above.

N Input parameter.

`N` is `INTEGER`

The order of the matrix `A`. `N` >= 0. When `N` = 0, DLANHS is set to zero.

A Input parameter.

`A` is `DOUBLE PRECISION`

`A` is an array, dimension (`LDA`, `N`). The n by n upper Hessenberg matrix A; the part of `A` below the first sub-diagonal is not referenced.

LDA Input parameter.

`LDA` is `INTEGER`

The leading dimension of the array `A`. `LDA` >= max(`N`,1).

WORK Output parameter.

`WORK` is `DOUBLE PRECISION`

`WORK` is an array, dimension (MAX(1,LWORK)),. where LWORK >= `N` when `NORM` = ‘I’; otherwise, `WORK` is not referenced.

Related Information

For this routine in other precisions, please see *clanhs*, *slanhs* and *zlanhs*.

4.17.267 dlansb

`dlansb` returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of an n by n symmetric band matrix A, with k super-diagonals.

Syntax

Fortran specification:

```
use armpl_library

double precision function dlansb(NORM, UPLO, N, K, AB, LDAB, WORK)
```

C specification:

```
#include "armpl.h"

double dlansb_(const char *norm, const char *uplo, const armpl_int_t *n,
               const armpl_int_t *k, const double *ab,
               const armpl_int_t *ldab, double *work, ... );
```

Returns

DLANSB = (max(abs(A(i,j))), NORM = ‘M’ or ‘m’

((norm1(A), NORM = ‘1’, ‘O’ or ‘o’ ((normI(A), NORM = ‘I’ or ‘i’ ((normF(A), NORM = ‘F’, ‘f’, ‘E’ or ‘e’

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

`NORM` is `CHARACTER*1`

Specifies the value to be returned in DLANSB as described above.

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

Specifies whether the upper or lower triangular part of the band matrix A is supplied. = ‘`U`’: Upper triangular part is supplied = ‘`L`’: Lower triangular part is supplied

N Input parameter.

`N` is `INTEGER`

The order of the matrix A. `N` >= 0. When `N` = 0, DLANSB is set to zero.

K Input parameter.

`K` is `INTEGER`

The number of super-diagonals or sub-diagonals of the band matrix A. `K` >= 0.

AB Input parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). The upper or lower triangle of the symmetric band matrix A, stored in the first K+1 rows of AB. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(k+1+i-j,j) = A(i,j) for max(1,j-k)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+k).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= K+1.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= N when NORM = 'I' or '1' or 'O'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *clansb*, *slansb* and *zlansb*.

4.17.268 dlansf

dlansf returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a real symmetric matrix A in RFP format.

Syntax

Fortran specification:

```
use armpl_library

double precision function dlansf(NORM, TRANSR, UPLO, N, A, WORK)
```

C specification:

```
#include "armpl.h"

double dlansf_(const char *norm, const char *transr, const char *uplo,
               const armpl_int_t *n, const double *a, double *work, ... );
```

Returns

DLANSF = (max(abs(A(i,j))), NORM = 'M' or 'm'

((norm1(A), NORM = '1', 'O' or 'o' ((normI(A), NORM = 'I' or 'i' ((normF(A), NORM = 'F', 'f', 'E' or 'e'

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in DLANSF as described above.

TRANSR Input parameter.

TRANSR is CHARACTER*1

Specifies whether the RFP format of A is normal or transposed format. = 'N': RFP format is Normal; = 'T': RFP format is Transpose.

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the RFP matrix A came from an upper or lower triangular matrix as follows: = 'U': RFP A came from an upper triangular matrix; = 'L': RFP A came from a lower triangular matrix.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0. When N = 0, DLANSF is set to zero.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (N*(N+1)/2);. On entry, the upper (if UPLO = 'U') or lower (if UPLO = 'L') part of the symmetric matrix A stored in RFP format. See the "Notes" below for more details. Unchanged on exit.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= N when NORM = 'I' or '1' or 'O'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *slansf*.

4.17.269 dlansp

dlansp returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a real symmetric matrix A, supplied in packed form.

Syntax

Fortran specification:

```
use armpl_library

double precision function dlansp(NORM, UPLO, N, AP, WORK)
```

C specification:

```
#include "armpl.h"

double dlansp_(const char *norm, const char *uplo, const armpl_int_t *n,
               const double *ap, double *work, ... );
```

Returns

DLANSP = (max(abs(A(i,j))), NORM = 'M' or 'm'

((norm1(A), NORM = '1', 'O' or 'o' ((normI(A), NORM = 'I' or 'i' ((normF(A), NORM = 'F', 'f', 'E' or 'e'

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in DLANSP as described above.

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is supplied. = 'U': Upper triangular part of A is supplied = 'L': Lower triangular part of A is supplied

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0. When N = 0, DLANSP is set to zero.

AP Input parameter.

AP is DOUBLE PRECISION

AP is an array, dimension (N*(N+1)/2). The upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= N when NORM = 'I' or '1' or 'O'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *clansp*, *slansp* and *zlansp*.

4.17.270 dlanst

dlanst returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a real symmetric tridiagonal matrix A.

Syntax

Fortran specification:

```
use armpl_library

double precision function dlanst(NORM, N, D, E)
```

C specification:

```
#include "armpl.h"

double dlanst_(const char *norm, const armpl_int_t *n, const double *d,
               const double *e, ... );
```

Returns

DLANST = (max(abs(A(i,j))), NORM = 'M' or 'm'

((norm1(A), NORM = '1', 'O' or 'o' ((normI(A), NORM = 'I' or 'i' ((normF(A), NORM = 'F', 'f', 'E' or 'e'

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in DLANST as described above.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0. When N = 0, DLANST is set to zero.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The diagonal elements of A.

E Input parameter.

E is DOUBLE PRECISION

E is an array, dimension (N-1). The (n-1) sub-diagonal or super-diagonal elements of A.

Related Information

For this routine in other precisions, please see *slanst*.

4.17.271 dlansy

dlansy returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a real symmetric matrix A.

Syntax

Fortran specification:

```
use armpl_library

double precision function dlansy(NORM, UPLO, N, A, LDA, WORK)
```

C specification:

```
#include "armpl.h"

double dlansy_(const char *norm, const char *uplo, const armpl_int_t *n,
               const double *a, const armpl_int_t *lda, double *work, ... );
```

Returns

DLANSY = (max(abs(A(i,j))), NORM = ‘M’ or ‘m’

((norm1(A), NORM = ‘1’, ‘O’ or ‘o’ ((normI(A), NORM = ‘I’ or ‘i’ ((normF(A), NORM = ‘F’, ‘f’, ‘E’ or ‘e’

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in DLANSY as described above.

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is to be referenced. = ‘U’: Upper triangular part of A is referenced = ‘L’: Lower triangular part of A is referenced

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0. When N = 0, DLANSY is set to zero.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). The symmetric matrix A. If UPLO = ‘U’, the leading n by n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = ‘L’, the leading n by n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(N,1).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= N when NORM = ‘I’ or ‘1’ or ‘O’; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *clansy*, *slansy* and *zlansy*. It also exists with a native C interface as *LAPACKE_dlansy*.

4.17.272 dlantb

`dlantb` returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of an n by n triangular band matrix A, with (k + 1) diagonals.

Syntax

Fortran specification:

```
use armpl_library

double precision function dlantb(NORM, UPLO, DIAG, N, K, AB, LDAB, WORK)
```

C specification:

```
#include "armpl.h"

double dlantb_(const char *norm, const char *uplo, const char *diag,
               const armpl_int_t *n, const armpl_int_t *k, const double *ab,
               const armpl_int_t *ldab, double *work, ... );
```

Returns

DLANTB = (max(abs(A(i,j))), NORM = ‘M’ or ‘m’

((norm1(A), NORM = ‘1’, ‘O’ or ‘o’ ((normI(A), NORM = ‘I’ or ‘i’ ((normF(A), NORM = ‘F’, ‘f’, ‘E’ or ‘e’

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

`NORM` is `CHARACTER*1`

Specifies the value to be returned in DLANTB as described above.

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

Specifies whether the matrix A is upper or lower triangular. = ‘`U`’: Upper triangular = ‘`L`’: Lower triangular

DIAG Input parameter.

`DIAG` is `CHARACTER*1`

Specifies whether or not the matrix A is unit triangular. = ‘`N`’: Non-unit triangular = ‘`U`’: Unit triangular

N Input parameter.

`N` is `INTEGER`

The order of the matrix A. `N` >= 0. When `N` = 0, DLANTB is set to zero.

K Input parameter.

`K` is `INTEGER`

The number of super-diagonals of the matrix A if `UPLO` = ‘`U`’, or the number of sub-diagonals of the matrix A if `UPLO` = ‘`L`’. `K` >= 0.

AB Input parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). The upper or lower triangular band matrix A, stored in the first k+1 rows of AB. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(k+1+i-j,j) = A(i,j) for max(1,j-k)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+k). Note that when DIAG = 'U', the elements of the array AB corresponding to the diagonal elements of the matrix A are not referenced, but are assumed to be one.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= K+1.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= N when NORM = 'I'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *clantb*, *slantb* and *zlantb*.

4.17.273 dlantp

dlantp returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a triangular matrix A, supplied in packed form.

Syntax

Fortran specification:

```
use armpl_library

double precision function dlantp(NORM, UPLO, DIAG, N, AP, WORK)
```

C specification:

```
#include "armpl.h"

double dlantp_(const char *norm, const char *uplo, const char *diag,
               const armpl_int_t *n, const double *ap, double *work, ... );
```

Returns

DLANTP = (max(abs(A(i,j))), NORM = 'M' or 'm'

((norm1(A), NORM = '1', 'O' or 'o' ((normI(A), NORM = 'I' or 'i' ((normF(A), NORM = 'F', 'f', 'E' or 'e'

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in DLANTP as described above.

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the matrix A is upper or lower triangular. = 'U': Upper triangular = 'L': Lower triangular

DIAG Input parameter.

DIAG is CHARACTER*1

Specifies whether or not the matrix A is unit triangular. = 'N': Non-unit triangular = 'U': Unit triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0. When N = 0, DLANTP is set to zero.

AP Input parameter.

AP is DOUBLE PRECISION

AP is an array, dimension (N*(N+1)/2). The upper or lower triangular matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n. Note that when DIAG = 'U', the elements of the array AP corresponding to the diagonal elements of the matrix A are not referenced, but are assumed to be one.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= N when NORM = 'I'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *clantp*, *slantp* and *zlantp*.

4.17.274 dlantr

dlantr returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a trapezoidal or triangular matrix A.

Syntax

Fortran specification:

```
use armpl_library

double precision function dlantr(NORM, UPLO, DIAG, M, N, A, LDA, WORK)
```

C specification:

```
#include "armpl.h"

double dlantr_(const char *norm, const char *uplo, const char *diag,
               const armpl_int_t *m, const armpl_int_t *n, const double *a,
               const armpl_int_t *lda, double *work, ... );
```

Returns

DLANTR = (max(abs(A(i,j))), NORM = ‘M’ or ‘m’

((norm1(A), NORM = ‘1’, ‘O’ or ‘o’ ((normI(A), NORM = ‘I’ or ‘i’ ((normF(A), NORM = ‘F’, ‘f’, ‘E’ or ‘e’

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in DLANTR as described above.

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the matrix A is upper or lower trapezoidal. = ‘U’: Upper trapezoidal = ‘L’: Lower trapezoidal Note that A is triangular instead of trapezoidal if M = N.

DIAG Input parameter.

DIAG is CHARACTER*1

Specifies whether or not the matrix A has unit diagonal. = ‘N’: Non-unit diagonal = ‘U’: Unit diagonal

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0, and if UPLO = ‘U’, M <= N. When M = 0, DLANTR is set to zero.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0, and if UPLO = ‘L’, N <= M. When N = 0, DLANTR is set to zero.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). The trapezoidal matrix A (A is triangular if M = N). If UPLO = ‘U’, the leading m by n upper trapezoidal part of the array A contains the upper trapezoidal matrix, and the strictly lower triangular part of A is not referenced. If UPLO = ‘L’, the leading m by n lower trapezoidal part of the array A contains the lower trapezoidal matrix, and the strictly upper triangular part of A is not referenced. Note that when DIAG = ‘U’, the diagonal elements of A are not referenced and are assumed to be one.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(M,1).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= M when NORM = 'I'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *clantr*, *slantr* and *zlantr*. It also exists with a native C interface as *LAPACKE_dlantr*.

4.17.275 dlanv2

dlanv2 computes the Schur factorization of a real 2-by-2 nonsymmetric matrix in standard form:

```
[ A   B ] = [ CS -SN ] [ AA   BB ] [ CS   SN ]
[ C   D ]   [ SN  CS ] [ CC   DD ] [-SN   CS ]
```

where either 1) CC = 0 so that AA and DD are real eigenvalues of the matrix, or 2) AA = DD and BB*CC < 0, so that AA + or - sqrt(BB*CC) are complex conjugate eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlanv2(A, B, C, D, RT1R, RT1I, RT2R, RT2I, CS, SN)
```

C specification:

```
#include "armpl.h"

void dlanv2_(double *a, double *b, double *c, double *d, double *rt1r,
             double *rt1i, double *rt2r, double *rt2i, double *cs,
             double *sn);
```

Parameters

A Input and output parameter.

A is DOUBLE PRECISION

B Input and output parameter.

B is DOUBLE PRECISION

C Input and output parameter.

C is DOUBLE PRECISION

D Input and output parameter.

D is DOUBLE PRECISION

On entry, the elements of the input matrix. On exit, they are overwritten by the elements of the standardised Schur form.

RT1R Output parameter.

RT1R is DOUBLE PRECISION

RT1I Output parameter.

RT1I is DOUBLE PRECISION

RT2R Output parameter.

RT2R is DOUBLE PRECISION

RT2I Output parameter.

RT2I is DOUBLE PRECISION

The real and imaginary parts of the eigenvalues. If the eigenvalues are a complex conjugate pair, RT1I > 0.

CS Output parameter.

CS is DOUBLE PRECISION

SN Output parameter.

SN is DOUBLE PRECISION

Parameters of the rotation matrix.

Related Information

For this routine in other precisions, please see *slanv2*.

4.17.276 dlapll

Given two column vectors X and Y, let

```
A = ( X Y ).
```

The subroutine first computes the QR factorization of A = Q*R, and then computes the SVD of the 2-by-2 upper triangular matrix R. The smaller singular value of R is returned in SSMIN, which is used as the measurement of the linear dependency of the vectors X and Y.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlapll(N, X, INCX, Y, INCY, SSMIN)
```

C specification:

```
#include "armpl.h"

void dlapll_(const armpl_int_t *n, double *x, const armpl_int_t *incx,
             double *y, const armpl_int_t *incy, double *ssmin);
```

Parameters

N Input parameter.

N is INTEGER

The length of the vectors X and Y.

X Input and output parameter.

X is DOUBLE PRECISION

X is an array,. dimension (1+(N-1)*INCX) On entry, X contains the N-vector X. On exit, X is overwritten.

INCX Input parameter.

INCX is INTEGER

The increment between successive elements of X. INCX > 0.

Y Input and output parameter.

Y is DOUBLE PRECISION

Y is an array,. dimension (1+(N-1)*INCY) On entry, Y contains the N-vector Y. On exit, Y is overwritten.

INCY Input parameter.

INCY is INTEGER

The increment between successive elements of Y. INCY > 0.

SSMIN Output parameter.

SSMIN is DOUBLE PRECISION

The smallest singular value of the N-by-2 matrix A = (X Y).

Related Information

For this routine in other precisions, please see *clapll*, *slapll* and *zlapll*.

4.17.277 dlapmr

dlapmr rearranges the rows of the M by N matrix X as specified by the permutation K(1),K(2),...,K(M) of the integers 1,...,M. If FORWRD = .TRUE., forward permutation:

```
X(K(I),*) is moved X(I,*) for I = 1,2,...,M.
```

If FORWRD = .FALSE., backward permutation:

```
X(I,*) is moved to X(K(I),*) for I = 1,2,...,M.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dlapmr(FORWRD, M, N, X, LDX, K)
```

C specification:

```
#include "armpl.h"

void dlapmr_(const armpl_int_t *forwrd, const armpl_int_t *m,
             const armpl_int_t *n, double *x, const armpl_int_t *ldx,
             armpl_int_t *k);
```

Parameters

FORWRD Input parameter.

FORWRD is LOGICAL

= .TRUE., forward permutation = .FALSE., backward permutation

M Input parameter.

M is INTEGER

The number of rows of the matrix X. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix X. N >= 0.

X Input and output parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, N). On entry, the M by N matrix X. On exit, X contains the permuted matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X, LDX >= MAX(1, M).

K Input and output parameter.

K is INTEGER array, dimension (M)

On entry, K contains the permutation vector. K is used as internal workspace, but reset to its original value on output.

Related Information

For this routine in other precisions, please see *clapmr*, *slapmr* and *zlapmr*. It also exists with a native C interface as *LAPACKE_dlapmr*.

4.17.278 dlapmt

dlapmt rearranges the columns of the M by N matrix X as specified by the permutation K(1),K(2),...,K(N) of the integers 1,...,N. If FORWRD = .TRUE., forward permutation:

```
X(*,K(J)) is moved X(*,J) for J = 1,2,...,N.
```

If FORWRD = .FALSE., backward permutation:

```
X(*,J) is moved to X(*,K(J)) for J = 1,2,...,N.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dlapmt(FORWRD, M, N, X, LDX, K)
```

C specification:

```
#include "armpl.h"

void dlapmt_(const armpl_int_t *forwrd, const armpl_int_t *m,
             const armpl_int_t *n, double *x, const armpl_int_t *ldx,
             armpl_int_t *k);
```

Parameters

FORWRD Input parameter.

FORWRD is LOGICAL

= .TRUE., forward permutation = .FALSE., backward permutation

M Input parameter.

M is INTEGER

The number of rows of the matrix X. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix X. N >= 0.

X Input and output parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, N). On entry, the M by N matrix X. On exit, X contains the permuted matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X, LDX >= MAX(1, M).

K Input and output parameter.

K is INTEGER array, dimension (N)

On entry, K contains the permutation vector. K is used as internal workspace, but reset to its original value on output.

Related Information

For this routine in other precisions, please see *clapmt*, *slapmt* and *zlapmt*. It also exists with a native C interface as *LAPACKE_dlapmt*.

4.17.279 dlapy2

dlapy2 returns sqrt(x**2+y**2), taking care not to cause unnecessary overflow.

Syntax

Fortran specification:

```
use armpl_library

double precision function dlapy2(X, Y)
```

C specification:

```
#include "armpl.h"

double dlapy2_(const double *x, const double *y);
```

Parameters

X Input parameter.

X is DOUBLE PRECISION

Y Input parameter.

Y is DOUBLE PRECISION

X and Y specify the values x and y.

Related Information

For this routine in other precisions, please see *slapy2*. It also exists with a native C interface as *LAPACKE_dlapy2*.

4.17.280 dlapy3

dlapy3 returns sqrt(x**2+y**2+z**2), taking care not to cause unnecessary overflow.

Syntax

Fortran specification:

```
use armpl_library

double precision function dlapy3(X, Y, Z)
```

C specification:

```
#include "armpl.h"

double dlapy3_(const double *x, const double *y, const double *z);
```

Parameters

X Input parameter.

X is DOUBLE PRECISION

Y Input parameter.

Y is DOUBLE PRECISION

Z Input parameter.

Z is DOUBLE PRECISION

X, Y and Z specify the values x, y and z.

Related Information

For this routine in other precisions, please see *slapy3*. It also exists with a native C interface as *LAPACKE_dlapy3*.

4.17.281 dlaqgb

`dlaqgb` equilibrates a general M by N band matrix A with KL subdiagonals and KU superdiagonals using the row and scaling factors in the vectors R and C.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlaqgb(M, N, KL, KU, AB, LDAB, R, C, ROWCND, COLCND, AMAX, EQUED)
```

C specification:

```
#include "armpl.h"

void dlaqgb_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *kl, const armpl_int_t *ku, double *ab,
             const armpl_int_t *ldab, const double *r, const double *c,
             const double *rowcnd, const double *colcnd, const double *amax,
             char *equed, ... );
```

Parameters

M Input parameter.

`M` is `INTEGER`

The number of rows of the matrix A. `M` >= 0.

N Input parameter.

`N` is `INTEGER`

The number of columns of the matrix A. `N` >= 0.

KL Input parameter.

`KL` is `INTEGER`

The number of subdiagonals within the band of A. `KL` >= 0.

KU Input parameter.

`KU` is `INTEGER`

The number of superdiagonals within the band of A. `KU` >= 0.

AB Input and output parameter.

`AB` is `DOUBLE PRECISION`

`AB` is an array, dimension (`LDAB`, `N`). On entry, the matrix A in band storage, in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array `AB` as follows: AB(ku+1+i-j,j) = A(i,j) for max(1,j-ku)<=i<=min(m,j+kl)

On exit, the equilibrated matrix, in the same storage format as A. See `EQUED` for the form of the equilibrated matrix.

LDAB Input parameter.

`LDAB` is `INTEGER`

The leading dimension of the array `AB`. LDA >= KL+KU+1.

R Input parameter.

R is DOUBLE PRECISION

R is an array, dimension (M). The row scale factors for A.

C Input parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). The column scale factors for A.

ROWCND Input parameter.

ROWCND is DOUBLE PRECISION

Ratio of the smallest R(i) to the largest R(i).

COLCND Input parameter.

COLCND is DOUBLE PRECISION

Ratio of the smallest C(i) to the largest C(i).

AMAX Input parameter.

AMAX is DOUBLE PRECISION

Absolute value of largest matrix entry.

EQUED Output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration = 'R': Row equilibration, i.e., A has been premultiplied by diag(R). = 'C': Column equilibration, i.e., A has been postmultiplied by diag(C). = 'B': Both row and column equilibration, i.e., A has been replaced by diag(R) * A * diag(C).

Related Information

For this routine in other precisions, please see *claqgb*, *slaqgb* and *zlaqgb*.

4.17.282 dlaqge

dlaqge equilibrates a general M by N matrix A using the row and column scaling factors in the vectors R and C.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlaqge(M, N, A, LDA, R, C, ROWCND, COLCND, AMAX, EQUED)
```

C specification:

```
#include "armpl.h"

void dlaqge_(const armpl_int_t *m, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, const double *r, const double *c,
             const double *rowcnd, const double *colcnd, const double *amax,
             char *equed, ... );
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the M by N matrix A. On exit, the equilibrated matrix. See EQUED for the form of the equilibrated matrix.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(M,1).

R Input parameter.

R is DOUBLE PRECISION

R is an array, dimension (M). The row scale factors for A.

C Input parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). The column scale factors for A.

ROWCND Input parameter.

ROWCND is DOUBLE PRECISION

Ratio of the smallest R(i) to the largest R(i).

COLCND Input parameter.

COLCND is DOUBLE PRECISION

Ratio of the smallest C(i) to the largest C(i).

AMAX Input parameter.

AMAX is DOUBLE PRECISION

Absolute value of largest matrix entry.

EQUED Output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration = 'R': Row equilibration, i.e., A has been premultiplied by diag(R). = 'C': Column equilibration, i.e., A has been postmultiplied by diag(C). = 'B': Both row and column equilibration, i.e., A has been replaced by diag(R) * A * diag(C).

Related Information

For this routine in other precisions, please see *claqge*, *slaqge* and *zlaqge*.

4.17.283 dlaqp2

`dlaqp2` computes a QR factorization with column pivoting of the block A(OFFSET+1:M,1:N). The block A(1:OFFSET,1:N) is accordingly pivoted, but not factorized.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlaqp2(M, N, OFFSET, A, LDA, JPVT, TAU, VN1, VN2, WORK)
```

C specification:

```
#include "armpl.h"

void dlaqp2_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *offset, double *a, const armpl_int_t *lda,
             armpl_int_t *jpvt, double *tau, double *vn1, double *vn2,
             double *work);
```

Parameters

M Input parameter.

`M` is `INTEGER`

The number of rows of the matrix `A`. `M` >= 0.

N Input parameter.

`N` is `INTEGER`

The number of columns of the matrix `A`. `N` >= 0.

OFFSET Input parameter.

`OFFSET` is `INTEGER`

The number of rows of the matrix `A` that must be pivoted but no factorized. `OFFSET` >= 0.

A Input and output parameter.

`A` is `DOUBLE PRECISION`

`A` is an array, dimension (`LDA`, `N`). On entry, the M-by-N matrix `A`. On exit, the upper triangle of block A(OFFSET+1:M,1:N) is the triangular factor obtained; the elements in block A(OFFSET+1:M,1:N) below the diagonal, together with the array `TAU`, represent the orthogonal matrix Q as a product of elementary reflectors. Block A(1:OFFSET,1:N) has been accordingly pivoted, but no factorized.

LDA Input parameter.

`LDA` is `INTEGER`

The leading dimension of the array `A`. `LDA` >= max(1, `M`).

JPVT Input and output parameter.

`JPVT` is `INTEGER` array, dimension (`N`)

On entry, if JPVT(i) .ne. 0, the i-th column of `A` is permuted to the front of `A`*P (a leading column); if JPVT(i) = 0, the i-th column of `A` is a free column. On exit, if JPVT(i) = k, then the i-th column of `A`*P was the k-th column of `A`.

TAU Output parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors.

VN1 Input and output parameter.

VN1 is DOUBLE PRECISION

VN1 is an array, dimension (N). The vector with the partial column norms.

VN2 Input and output parameter.

VN2 is DOUBLE PRECISION

VN2 is an array, dimension (N). The vector with the exact column norms.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (N) .

Related Information

For this routine in other precisions, please see *claqp2*, *slaqp2* and *zlaqp2*.

4.17.284 dlaqps

dlaqps computes a step of QR factorization with column pivoting of a real M-by-N matrix A by using Blas-3. It tries to factorize NB columns from A starting from the row OFFSET+1, and updates all of the matrix with Blas-3 xGEMM.

In some cases, due to catastrophic cancellations, it cannot factorize NB columns. Hence, the actual number of factorized columns is returned in KB.

Block A(1:OFFSET,1:N) is accordingly pivoted, but not factorized.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlaqps(M, N, OFFSET, NB, KB, A, LDA, JPVT, TAU, VN1, VN2, AUXV, F,
                  LDF)
```

C specification:

```
#include "armpl.h"

void dlaqps_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *offset, const armpl_int_t *nb,
             armpl_int_t *kb, double *a, const armpl_int_t *lda,
             armpl_int_t *jpvt, double *tau, double *vn1, double *vn2,
             double *auxv, double *f, const armpl_int_t *ldf);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0

OFFSET Input parameter.

OFFSET is INTEGER

The number of rows of A that have been factorized in previous steps.

NB Input parameter.

NB is INTEGER

The number of columns to factorize.

KB Output parameter.

KB is INTEGER

The number of columns actually factorized.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, block A(OFFSET+1:M,1:KB) is the triangular factor obtained and block A(1:OFFSET,1:N) has been accordingly pivoted, but no factorized. The rest of the matrix, block A(OFFSET+1:M,KB+1:N) has been updated.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

JPVT Input and output parameter.

JPVT is INTEGER array, dimension (N)

JPVT(I) = K <==> Column K of the full matrix A has been permuted into position I in AP.

TAU Output parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (KB). The scalar factors of the elementary reflectors.

VN1 Input and output parameter.

VN1 is DOUBLE PRECISION

VN1 is an array, dimension (N). The vector with the partial column norms.

VN2 Input and output parameter.

VN2 is DOUBLE PRECISION

VN2 is an array, dimension (N). The vector with the exact column norms.

AUXV Input and output parameter.

AUXV is DOUBLE PRECISION

AUXV is an array, dimension (NB). Auxiliar vector.

F Input and output parameter.

F is DOUBLE PRECISION

F is an array, dimension (LDF, NB). Matrix $F^T = L*Y^T *A$.

LDF Input parameter.

LDF is INTEGER

The leading dimension of the array F. LDF >= max(1, N).

Related Information

For this routine in other precisions, please see *claqps*, *slaqps* and *zlaqps*.

4.17.285 dlaqr0

```
DLAQR0 computes the eigenvalues of a Hessenberg matrix H
   and, optionally, the matrices T and Z from the Schur decomposition
   H = Z T Z**T, where T is an upper quasi-triangular matrix (the
   Schur form), and Z is the orthogonal matrix of Schur vectors.

   Optionally Z may be postmultiplied into an input orthogonal
   matrix Q so that this routine can give the Schur factorization
   of a matrix A which has been reduced to the Hessenberg form H
   by the orthogonal matrix Q:  A = Q*H*Q**T = (QZ)*T*(QZ)**T.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dlaqr0(WANTT, WANTZ, N, ILO, IHI, H, LDH, WR, WI, ILOZ, IHIZ, Z,
                  LDZ, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dlaqr0_(const armpl_int_t *wantt, const armpl_int_t *wantz,
             const armpl_int_t *n, const armpl_int_t *ilo,
             const armpl_int_t *ihi, double *h, const armpl_int_t *ldh,
             double *wr, double *wi, const armpl_int_t *iloz,
             const armpl_int_t *ihiz, double *z, const armpl_int_t *ldz,
             double *work, const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

WANTT Input parameter.

WANTT is LOGICAL

= .TRUE. : the full Schur form T is required; = .FALSE.: only eigenvalues are required.

WANTZ Input parameter.

WANTZ is LOGICAL

= .TRUE. : the matrix of Schur vectors Z is required; = .FALSE.: Schur vectors are not required.

N Input parameter.

N is INTEGER

The order of the matrix H. N .GE. 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

It is assumed that H is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N and, if ILO.GT.1, H(ILO,ILO-1) is zero. ILO and IHI are normally set by a previous call to DGEBAL, and then passed to DGEHRD when the matrix output by DGEBAL is reduced to Hessenberg form. Otherwise, ILO and IHI should be set to 1 and N, respectively. If N.GT.0, then 1.LE.ILO.LE.IHI.LE.N. If N = 0, then ILO = 1 and IHI = 0.

H Input and output parameter.

H is DOUBLE PRECISION

H is an array, dimension (LDH, N). On entry, the upper Hessenberg matrix H. On exit, if INFO = 0 and WANTT is .TRUE., then H contains the upper quasi-triangular matrix T from the Schur decomposition (the Schur form); 2-by-2 diagonal blocks (corresponding to complex conjugate pairs of eigenvalues) are returned in standard form, with H(i,i) = H(i+1,i+1) and H(i+1,i)*H(i,i+1).LT.0. If INFO = 0 and WANTT is .FALSE., then the contents of H are unspecified on exit. (The output value of H when INFO.GT.0 is given under the description of INFO below.)

This subroutine may explicitly set H(i,j) = 0 for i.GT.j and j = 1, 2, ... ILO-1 or j = IHI+1, IHI+2, ... N.

LDH Input parameter.

LDH is INTEGER

The leading dimension of the array H. LDH .GE. max(1, N).

WR Output parameter.

WR is DOUBLE PRECISION

WR is an array, dimension (IHI) .

WI Output parameter.

WI is DOUBLE PRECISION

WI is an array, dimension (IHI). The real and imaginary parts, respectively, of the computed eigenvalues of H(ILO:IHI,ILO:IHI) are stored in WR(ILO:IHI) and WI(ILO:IHI). If two eigenvalues are computed as a complex conjugate pair, they are stored in consecutive elements of WR and WI, say the i-th and (i+1)th, with WI(i) .GT. 0 and WI(i+1) .LT. 0. If WANTT is .TRUE., then the eigenvalues are stored in the same order as on the diagonal of the Schur form returned in H, with WR(i) = H(i,i) and, if H(i:i+1,i:i+1) is a 2-by-2 diagonal block, WI(i) = sqrt(-H(i+1,i)*H(i,i+1)) and WI(i+1) = -WI(i).

ILOZ Input parameter.

ILOZ is INTEGER

IHIZ Input parameter.

IHIZ is INTEGER

Specify the rows of Z to which transformations must be applied if WANTZ is .TRUE.. 1 .LE. ILOZ .LE. ILO; IHI .LE. IHIZ .LE. N.

Z Input and output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, IHI). If WANTZ is .FALSE., then Z is not referenced. If WANTZ is .TRUE., then Z(ILO:IHI,ILOZ:IHIZ) is replaced by Z(ILO:IHI,ILOZ:IHIZ)*U where U is the orthogonal Schur factor of H(ILO:IHI,ILO:IHI). (The output value of Z when INFO.GT.0 is given under the description of INFO below.)

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. if WANTZ is .TRUE. then LDZ.GE.MAX(1, IHIZ). Otherwize, LDZ.GE.1.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension LWORK. On exit, if LWORK = -1, WORK(1) returns an estimate of the optimal value for LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK .GE. max(1, N) is sufficient, but LWORK typically as large as 6* N may be required for optimal performance. A workspace query to determine the optimal workspace size is recommended.

If LWORK = -1, then DLAQR0 does a workspace query. In this case, DLAQR0 checks the input parameters and estimates the optimal workspace size for the given values of N, ILO and IHI. The estimate is returned in WORK(1). No error message related to LWORK is issued by XERBLA. Neither H nor Z are accessed.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit .GT. 0: if INFO = i, DLAQR0 failed to compute all of the eigenvalues. Elements 1:ilo-1 and i+1:n of WR and WI contain those eigenvalues which have been successfully computed. (Failures are rare.)

If INFO .GT. 0 and WANT is .FALSE., then on exit, the remaining unconverged eigenvalues are the eigenvalues of the upper Hessenberg matrix rows and columns ILO through INFO of the final, output value of H.

If INFO .GT. 0 and WANTT is .TRUE., then on exit

(*) (initial value of H)*U = U*(final value of H)

where U is an orthogonal matrix. The final value of H is upper Hessenberg and quasi-triangular in rows and columns INFO+1 through IHI.

If INFO .GT. 0 and WANTZ is .TRUE., then on exit

(final value of Z(ILO:IHI,ILOZ:IHIZ) = (initial value of Z(ILO:IHI,ILOZ:IHIZ)*U

where U is the orthogonal matrix in (*) (regard- less of the value of WANTT.)

If INFO .GT. 0 and WANTZ is .FALSE., then Z is not accessed.

Related Information

For this routine in other precisions, please see *claqr0*, *slaqr0* and *zlaqr0*.

4.17.286 dlaqr1

```
Given a 2-by-2 or 3-by-3 matrix H, DLAQR1 sets v to a
     scalar multiple of the first column of the product

     (*)  K = (H - (sr1 + i*si1)*I)*(H - (sr2 + i*si2)*I)

     scaling to avoid overflows and most underflows. It
     is assumed that either

               1) sr1 = sr2 and si1 = -si2
          or
               2) si1 = si2 = 0.

     This is useful for starting double implicit shift bulges
     in the QR algorithm.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dlaqr1(N, H, LDH, SR1, SI1, SR2, SI2, V)
```

C specification:

```
#include "armpl.h"

void dlaqr1_(const armpl_int_t *n, const double *h, const armpl_int_t *ldh,
             const double *sr1, double *si1, double *sr2, double *si2,
             double *v);
```

Parameters

N Input parameter.

N is INTEGER

Order of the matrix H. N must be either 2 or 3.

H Input parameter.

H is DOUBLE PRECISION

H is an array, dimension (LDH, N). The 2-by-2 or 3-by-3 matrix H in (*).

LDH Input parameter.

LDH is INTEGER

The leading dimension of H as declared in the calling procedure. LDH.GE.N

SR1 Input parameter.

SR1 is DOUBLE PRECISION

SI1 Input parameter.

SI1 is DOUBLE PRECISION

SR2 Input parameter.

SR2 is DOUBLE PRECISION

SI2 Input parameter.

SI2 is DOUBLE PRECISION

The shifts in (*).

V Output parameter.

V is DOUBLE PRECISION

V is an array, dimension (N). A scalar multiple of the first column of the matrix K in (*).

Related Information

For this routine in other precisions, please see *claqr1*, *slaqr1* and *zlaqr1*.

4.17.287 dlaqr2

```
DLAQR2 is identical to DLAQR3 except that it avoids
   recursion by calling DLAHQR instead of DLAQR4.

   Aggressive early deflation:

   This subroutine accepts as input an upper Hessenberg matrix
   H and performs an orthogonal similarity transformation
   designed to detect and deflate fully converged eigenvalues from
   a trailing principal submatrix.  On output H has been over-
   written by a new Hessenberg matrix that is a perturbation of
   an orthogonal similarity transformation of H.  It is to be
   hoped that the final version of H has many zero subdiagonal
   entries.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dlaqr2(WANTT, WANTZ, N, KTOP, KBOT, NW, H, LDH, ILOZ, IHIZ, Z, LDZ,
                  NS, ND, SR, SI, V, LDV, NH, T, LDT, NV, WV, LDWV, WORK,
                  LWORK)
```

C specification:

```
#include "armpl.h"

void dlaqr2_(const armpl_int_t *wantt, const armpl_int_t *wantz,
             const armpl_int_t *n, const armpl_int_t *ktop,
             const armpl_int_t *kbot, const armpl_int_t *nw, double *h,
             const armpl_int_t *ldh, const armpl_int_t *iloz,
             const armpl_int_t *ihiz, double *z, const armpl_int_t *ldz,
             armpl_int_t *ns, armpl_int_t *nd, double *sr, double *si,
             double *v, const armpl_int_t *ldv, const armpl_int_t *nh,
             double *t, const armpl_int_t *ldt, const armpl_int_t *nv,
             double *wv, const armpl_int_t *ldwv, double *work,
             const armpl_int_t *lwork);
```

Parameters

WANTT Input parameter.

WANTT is LOGICAL

If .TRUE., then the Hessenberg matrix H is fully updated so that the quasi-triangular Schur factor may be computed (in cooperation with the calling subroutine). If .FALSE., then only enough of H is updated to preserve the eigenvalues.

WANTZ Input parameter.

WANTZ is LOGICAL

If .TRUE., then the orthogonal matrix Z is updated so so that the orthogonal Schur factor may be computed (in cooperation with the calling subroutine). If .FALSE., then Z is not referenced.

N Input parameter.

N is INTEGER

The order of the matrix H and (if WANTZ is .TRUE.) the order of the orthogonal matrix Z.

KTOP Input parameter.

KTOP is INTEGER

It is assumed that either KTOP = 1 or H(KTOP,KTOP-1)=0. KBOT and KTOP together determine an isolated block along the diagonal of the Hessenberg matrix.

KBOT Input parameter.

KBOT is INTEGER

It is assumed without a check that either KBOT = N or H(KBOT+1, KBOT)=0. KBOT and KTOP together determine an isolated block along the diagonal of the Hessenberg matrix.

NW Input parameter.

NW is INTEGER

Deflation window size. 1 .LE. NW .LE. (KBOT-KTOP+1).

H Input and output parameter.

H is DOUBLE PRECISION

H is an array, dimension (LDH, N). On input the initial N-by-N section of H stores the Hessenberg matrix undergoing aggressive early deflation. On output H has been transformed by an orthogonal similarity transformation, perturbed, and the returned to Hessenberg form that (it is to be hoped) has some zero subdiagonal entries.

LDH Input parameter.

LDH is INTEGER

Leading dimension of H just as declared in the calling subroutine. N .LE. LDH

ILOZ Input parameter.

ILOZ is INTEGER

IHIZ Input parameter.

IHIZ is INTEGER

Specify the rows of Z to which transformations must be applied if WANTZ is .TRUE.. 1 .LE. ILOZ .LE. IHIZ .LE. N.

Z Input and output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, N). IF WANTZ is .TRUE., then on output, the orthogonal similarity transformation mentioned above has been accumulated into Z(ILOZ:IHIZ,ILOZ:IHIZ) from the right. If WANTZ is .FALSE., then Z is unreferenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of Z just as declared in the calling subroutine. 1 .LE. LDZ.

NS Output parameter.

NS is INTEGER

The number of unconverged (ie approximate) eigenvalues returned in SR and SI that may be used as shifts by the calling subroutine.

ND Output parameter.

ND is INTEGER

The number of converged eigenvalues uncovered by this subroutine.

SR Output parameter.

SR is DOUBLE PRECISION

SR is an array, dimension (KBOT) .

SI Output parameter.

SI is DOUBLE PRECISION

SI is an array, dimension (KBOT). On output, the real and imaginary parts of approximate eigenvalues that may be used for shifts are stored in SR(KBOT-ND-NS+1) through SR(KBOT-ND) and SI(KBOT-ND-NS+1) through SI(KBOT-ND), respectively. The real and imaginary parts of converged eigenvalues are stored in SR(KBOT-ND+1) through SR(KBOT) and SI(KBOT-ND+1) through SI(KBOT), respectively.

V Output parameter.

V is DOUBLE PRECISION

V is an array, dimension (LDV, NW). An NW-by-NW work array.

LDV Input parameter.

LDV is INTEGER

The leading dimension of V just as declared in the calling subroutine. NW .LE. LDV

NH Input parameter.

NH is INTEGER

The number of columns of T. NH.GE.NW.

T Output parameter.

T is DOUBLE PRECISION

T is an array, dimension (LDT, NW) .

LDT Input parameter.

LDT is INTEGER

The leading dimension of T just as declared in the calling subroutine. NW .LE. LDT

NV Input parameter.

NV is INTEGER

The number of rows of work array WV available for workspace. NV.GE.NW.

WV Output parameter.

WV is DOUBLE PRECISION

WV is an array, dimension (LDWV, NW) .

LDWV Input parameter.

LDWV is INTEGER

The leading dimension of W just as declared in the calling subroutine. NW .LE. LDV

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (LWORK). On exit, WORK(1) is set to an estimate of the optimal value of LWORK for the given values of N, NW, KTOP and KBOT.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the work array WORK. LWORK = 2*NW suffices, but greater efficiency may result from larger values of LWORK.

If LWORK = -1, then a workspace query is assumed; DLAQR2 only estimates the optimal workspace size for the given values of N, NW, KTOP and KBOT. The estimate is returned in WORK(1). No error message related to LWORK is issued by XERBLA. Neither H nor Z are accessed.

Related Information

For this routine in other precisions, please see *claqr2*, *slaqr2* and *zlaqr2*.

4.17.288 dlaqr3

```
Aggressive early deflation:

   DLAQR3 accepts as input an upper Hessenberg matrix
   H and performs an orthogonal similarity transformation
   designed to detect and deflate fully converged eigenvalues from
   a trailing principal submatrix.  On output H has been over-
   written by a new Hessenberg matrix that is a perturbation of
   an orthogonal similarity transformation of H.  It is to be
   hoped that the final version of H has many zero subdiagonal
   entries.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dlaqr3(WANTT, WANTZ, N, KTOP, KBOT, NW, H, LDH, ILOZ, IHIZ, Z, LDZ,
                  NS, ND, SR, SI, V, LDV, NH, T, LDT, NV, WV, LDWV, WORK,
                  LWORK)
```

C specification:

```
#include "armpl.h"

void dlaqr3_(const armpl_int_t *wantt, const armpl_int_t *wantz,
             const armpl_int_t *n, const armpl_int_t *ktop,
             const armpl_int_t *kbot, const armpl_int_t *nw, double *h,
             const armpl_int_t *ldh, const armpl_int_t *iloz,
             const armpl_int_t *ihiz, double *z, const armpl_int_t *ldz,
             armpl_int_t *ns, armpl_int_t *nd, double *sr, double *si,
             double *v, const armpl_int_t *ldv, const armpl_int_t *nh,
             double *t, const armpl_int_t *ldt, const armpl_int_t *nv,
             double *wv, const armpl_int_t *ldwv, double *work,
             const armpl_int_t *lwork);
```

Parameters

WANTT Input parameter.

WANTT is LOGICAL

If .TRUE., then the Hessenberg matrix H is fully updated so that the quasi-triangular Schur factor may be computed (in cooperation with the calling subroutine). If .FALSE., then only enough of H is updated to preserve the eigenvalues.

WANTZ Input parameter.

WANTZ is LOGICAL

If .TRUE., then the orthogonal matrix Z is updated so so that the orthogonal Schur factor may be computed (in cooperation with the calling subroutine). If .FALSE., then Z is not referenced.

N Input parameter.

N is INTEGER

The order of the matrix H and (if WANTZ is .TRUE.) the order of the orthogonal matrix Z.

KTOP Input parameter.

KTOP is INTEGER

It is assumed that either KTOP = 1 or H(KTOP,KTOP-1)=0. KBOT and KTOP together determine an isolated block along the diagonal of the Hessenberg matrix.

KBOT Input parameter.

KBOT is INTEGER

It is assumed without a check that either KBOT = N or H(KBOT+1, KBOT)=0. KBOT and KTOP together determine an isolated block along the diagonal of the Hessenberg matrix.

NW Input parameter.

NW is INTEGER

Deflation window size. 1 .LE. NW .LE. (KBOT-KTOP+1).

H Input and output parameter.

H is DOUBLE PRECISION

H is an array, dimension (LDH, N). On input the initial N-by-N section of H stores the Hessenberg matrix undergoing aggressive early deflation. On output H has been transformed by an orthogonal similarity transformation, perturbed, and the returned to Hessenberg form that (it is to be hoped) has some zero subdiagonal entries.

LDH Input parameter.

LDH is INTEGER

Leading dimension of H just as declared in the calling subroutine. N .LE. LDH

ILOZ Input parameter.

ILOZ is INTEGER

IHIZ Input parameter.

IHIZ is INTEGER

Specify the rows of Z to which transformations must be applied if WANTZ is .TRUE.. 1 .LE. ILOZ .LE. IHIZ .LE. N.

Z Input and output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, N). IF WANTZ is .TRUE., then on output, the orthogonal similarity transformation mentioned above has been accumulated into Z(ILOZ:IHIZ,ILOZ:IHIZ) from the right. If WANTZ is .FALSE., then Z is unreferenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of Z just as declared in the calling subroutine. 1 .LE. LDZ.

NS Output parameter.

NS is INTEGER

The number of unconverged (ie approximate) eigenvalues returned in SR and SI that may be used as shifts by the calling subroutine.

ND Output parameter.

ND is INTEGER

The number of converged eigenvalues uncovered by this subroutine.

SR Output parameter.

SR is DOUBLE PRECISION

SR is an array, dimension (KBOT) .

SI Output parameter.

SI is DOUBLE PRECISION

SI is an array, dimension (KBOT). On output, the real and imaginary parts of approximate eigenvalues that may be used for shifts are stored in SR(KBOT-ND-NS+1) through SR(KBOT-ND) and SI(KBOT-ND-NS+1) through SI(KBOT-ND), respectively. The real and imaginary parts of converged eigenvalues are stored in SR(KBOT-ND+1) through SR(KBOT) and SI(KBOT-ND+1) through SI(KBOT), respectively.

V Output parameter.

V is DOUBLE PRECISION

V is an array, dimension (LDV, NW). An NW-by-NW work array.

LDV Input parameter.

LDV is INTEGER

The leading dimension of V just as declared in the calling subroutine. NW .LE. LDV

NH Input parameter.

NH is INTEGER

The number of columns of T. NH.GE.NW.

T Output parameter.

T is DOUBLE PRECISION

T is an array, dimension (LDT, NW) .

LDT Input parameter.

LDT is INTEGER

The leading dimension of T just as declared in the calling subroutine. NW .LE. LDT

NV Input parameter.

NV is INTEGER

The number of rows of work array WV available for workspace. NV.GE.NW.

WV Output parameter.

WV is DOUBLE PRECISION

WV is an array, dimension (LDWV, NW) .

LDWV Input parameter.

LDWV is INTEGER

The leading dimension of W just as declared in the calling subroutine. NW .LE. LDV

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (LWORK). On exit, WORK(1) is set to an estimate of the optimal value of LWORK for the given values of N, NW, KTOP and KBOT.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the work array WORK. LWORK = 2*NW suffices, but greater efficiency may result from larger values of LWORK.

If LWORK = -1, then a workspace query is assumed; DLAQR3 only estimates the optimal workspace size for the given values of N, NW, KTOP and KBOT. The estimate is returned in WORK(1). No error message related to LWORK is issued by XERBLA. Neither H nor Z are accessed.

Related Information

For this routine in other precisions, please see *claqr3*, *slaqr3* and *zlaqr3*.

4.17.289 dlaqr4

```
DLAQR4 implements one level of recursion for DLAQR0.
   It is a complete implementation of the small bulge multi-shift
   QR algorithm.  It may be called by DLAQR0 and, for large enough
   deflation window size, it may be called by DLAQR3.  This
   subroutine is identical to DLAQR0 except that it calls DLAQR2
   instead of DLAQR3.

   DLAQR4 computes the eigenvalues of a Hessenberg matrix H
   and, optionally, the matrices T and Z from the Schur decomposition
   H = Z T Z**T, where T is an upper quasi-triangular matrix (the
   Schur form), and Z is the orthogonal matrix of Schur vectors.

   Optionally Z may be postmultiplied into an input orthogonal
```

(continues on next page)

(continued from previous page)

```
   matrix Q so that this routine can give the Schur factorization
   of a matrix A which has been reduced to the Hessenberg form H
   by the orthogonal matrix Q:  A = Q*H*Q**T = (QZ)*T*(QZ)**T.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dlaqr4(WANTT, WANTZ, N, ILO, IHI, H, LDH, WR, WI, ILOZ, IHIZ, Z,
                  LDZ, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dlaqr4_(const armpl_int_t *wantt, const armpl_int_t *wantz,
             const armpl_int_t *n, const armpl_int_t *ilo,
             const armpl_int_t *ihi, double *h, const armpl_int_t *ldh,
             double *wr, double *wi, const armpl_int_t *iloz,
             const armpl_int_t *ihiz, double *z, const armpl_int_t *ldz,
             double *work, const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

WANTT Input parameter.

WANTT is LOGICAL

= .TRUE. : the full Schur form T is required; = .FALSE.: only eigenvalues are required.

WANTZ Input parameter.

WANTZ is LOGICAL

= .TRUE. : the matrix of Schur vectors Z is required; = .FALSE.: Schur vectors are not required.

N Input parameter.

N is INTEGER

The order of the matrix H. N .GE. 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

It is assumed that H is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N and, if ILO.GT.1, H(ILO,ILO-1) is zero. ILO and IHI are normally set by a previous call to DGEBAL, and then passed to DGEHRD when the matrix output by DGEBAL is reduced to Hessenberg form. Otherwise, ILO and IHI should be set to 1 and N, respectively. If N.GT.0, then 1.LE.ILO.LE.IHI.LE.N. If N = 0, then ILO = 1 and IHI = 0.

H Input and output parameter.

H is DOUBLE PRECISION

H is an array, dimension (LDH, N). On entry, the upper Hessenberg matrix H. On exit, if INFO = 0 and WANTT is .TRUE., then H contains the upper quasi-triangular matrix T from the Schur decomposition (the Schur

form); 2-by-2 diagonal blocks (corresponding to complex conjugate pairs of eigenvalues) are returned in standard form, with H(i,i) = H(i+1,i+1) and H(i+1,i)*H(i,i+1).LT.0. If INFO = 0 and WANTT is .FALSE., then the contents of H are unspecified on exit. (The output value of H when INFO.GT.0 is given under the description of INFO below.)

This subroutine may explicitly set H(i,j) = 0 for i.GT.j and j = 1, 2, ... ILO-1 or j = IHI+1, IHI+2, ... N.

LDH Input parameter.

LDH is INTEGER

The leading dimension of the array H. LDH .GE. max(1, N).

WR Output parameter.

WR is DOUBLE PRECISION

WR is an array, dimension (IHI) .

WI Output parameter.

WI is DOUBLE PRECISION

WI is an array, dimension (IHI). The real and imaginary parts, respectively, of the computed eigenvalues of H(ILO:IHI,ILO:IHI) are stored in WR(ILO:IHI) and WI(ILO:IHI). If two eigenvalues are computed as a complex conjugate pair, they are stored in consecutive elements of WR and WI, say the i-th and (i+1)th, with WI(i) .GT. 0 and WI(i+1) .LT. 0. If WANTT is .TRUE., then the eigenvalues are stored in the same order as on the diagonal of the Schur form returned in H, with WR(i) = H(i,i) and, if H(i:i+1,i:i+1) is a 2-by-2 diagonal block, WI(i) = sqrt(-H(i+1,i)*H(i,i+1)) and WI(i+1) = -WI(i).

ILOZ Input parameter.

ILOZ is INTEGER

IHIZ Input parameter.

IHIZ is INTEGER

Specify the rows of Z to which transformations must be applied if WANTZ is .TRUE.. 1 .LE. ILOZ .LE. ILO; IHI .LE. IHIZ .LE. N.

Z Input and output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, IHI). If WANTZ is .FALSE., then Z is not referenced. If WANTZ is .TRUE., then Z(ILO:IHI,ILOZ:IHIZ) is replaced by Z(ILO:IHI,ILOZ:IHIZ)*U where U is the orthogonal Schur factor of H(ILO:IHI,ILO:IHI). (The output value of Z when INFO.GT.0 is given under the description of INFO below.)

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. if WANTZ is .TRUE. then LDZ.GE.MAX(1, IHIZ). Otherwize, LDZ.GE.1.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension LWORK. On exit, if LWORK = -1, WORK(1) returns an estimate of the optimal value for LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK .GE. max(1, N) is sufficient, but LWORK typically as large as 6* N may be required for optimal performance. A workspace query to determine the optimal workspace size is recommended.

If LWORK = -1, then DLAQR4 does a workspace query. In this case, DLAQR4 checks the input parameters and estimates the optimal workspace size for the given values of N, ILO and IHI. The estimate is returned in WORK(1). No error message related to LWORK is issued by XERBLA. Neither H nor Z are accessed.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit .GT. 0: if INFO = i, DLAQR4 failed to compute all of the eigenvalues. Elements 1:ilo-1 and i+1:n of WR and WI contain those eigenvalues which have been successfully computed. (Failures are rare.)

If INFO .GT. 0 and WANT is .FALSE., then on exit, the remaining unconverged eigenvalues are the eigenvalues of the upper Hessenberg matrix rows and columns ILO through INFO of the final, output value of H.

If INFO .GT. 0 and WANTT is .TRUE., then on exit

(*) (initial value of H)*U = U*(final value of H)

where U is a orthogonal matrix. The final value of H is upper Hessenberg and triangular in rows and columns INFO+1 through IHI.

If INFO .GT. 0 and WANTZ is .TRUE., then on exit

(final value of Z(ILO:IHI,ILOZ:IHIZ) = (initial value of Z(ILO:IHI,ILOZ:IHIZ)*U

where U is the orthogonal matrix in (*) (regard- less of the value of WANTT.)

If INFO .GT. 0 and WANTZ is .FALSE., then Z is not accessed.

Related Information

For this routine in other precisions, please see *claqr4*, *slaqr4* and *zlaqr4*.

4.17.290 dlaqr5

```
DLAQR5, called by DLAQR0, performs a
   single small-bulge multi-shift QR sweep.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dlaqr5(WANTT, WANTZ, KACC22, N, KTOP, KBOT, NSHFTS, SR, SI, H, LDH,
                  ILOZ, IHIZ, Z, LDZ, V, LDV, U, LDU, NV, WV, LDWV, NH, WH,
                  LDWH)
```

C specification:

```
#include "armpl.h"

void dlaqr5_(const armpl_int_t *wantt, const armpl_int_t *wantz,
             const armpl_int_t *kacc22, const armpl_int_t *n,
             const armpl_int_t *ktop, const armpl_int_t *kbot,
             const armpl_int_t *nshfts, double *sr, double *si, double *h,
             const armpl_int_t *ldh, const armpl_int_t *iloz,
             const armpl_int_t *ihiz, double *z, const armpl_int_t *ldz,
             double *v, const armpl_int_t *ldv, double *u,
             const armpl_int_t *ldu, const armpl_int_t *nv, double *wv,
```

(continues on next page)

(continued from previous page)

```
                const armpl_int_t *ldwv, const armpl_int_t *nh, double *wh,
                const armpl_int_t *ldwh);
```

Parameters

WANTT Input parameter.

WANTT is LOGICAL

WANTT = .true. if the quasi-triangular Schur factor is being computed. WANTT is set to .false. otherwise.

WANTZ Input parameter.

WANTZ is LOGICAL

WANTZ = .true. if the orthogonal Schur factor is being computed. WANTZ is set to .false. otherwise.

KACC22 Input parameter.

KACC22 is INTEGER with value 0, 1, or 2.

Specifies the computation mode of far-from-diagonal orthogonal updates. = 0: DLAQR5 does not accumulate reflections and does not use matrix-matrix multiply to update far-from-diagonal matrix entries. = 1: DLAQR5 accumulates reflections and uses matrix-matrix multiply to update the far-from-diagonal matrix entries. = 2: DLAQR5 accumulates reflections, uses matrix-matrix multiply to update the far-from-diagonal matrix entries, and takes advantage of 2-by-2 block structure during matrix multiplies.

N Input parameter.

N is INTEGER

N is the order of the Hessenberg matrix H upon which this subroutine operates.

KTOP Input parameter.

KTOP is INTEGER

KBOT Input parameter.

KBOT is INTEGER

These are the first and last rows and columns of an isolated diagonal block upon which the QR sweep is to be applied. It is assumed without a check that either KTOP = 1 or H(KTOP,KTOP-1) = 0 and either KBOT = N or H(KBOT+1, KBOT) = 0.

NSHFTS Input parameter.

NSHFTS is INTEGER

NSHFTS gives the number of simultaneous shifts. NSHFTS must be positive and even.

SR Input and output parameter.

SR is DOUBLE PRECISION

SR is an array, dimension (NSHFTS) .

SI Input and output parameter.

SI is DOUBLE PRECISION

SI is an array, dimension (NSHFTS). SR contains the real parts and SI contains the imaginary parts of the NSHFTS shifts of origin that define the multi-shift QR sweep. On output SR and SI may be reordered.

H Input and output parameter.

H is DOUBLE PRECISION

H is an array, dimension (LDH, N). On input H contains a Hessenberg matrix. On output a multi-shift QR sweep with shifts SR(J)+i*SI(J) is applied to the isolated diagonal block in rows and columns KTOP through KBOT.

LDH Input parameter.

LDH is INTEGER

LDH is the leading dimension of H just as declared in the calling procedure. LDH.GE.MAX(1, N).

ILOZ Input parameter.

ILOZ is INTEGER

IHIZ Input parameter.

IHIZ is INTEGER

Specify the rows of Z to which transformations must be applied if WANTZ is .TRUE.. 1 .LE. ILOZ .LE. IHIZ .LE. N

Z Input and output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, IHIZ). If WANTZ = .TRUE., then the QR Sweep orthogonal similarity transformation is accumulated into Z(ILOZ:IHIZ,ILOZ:IHIZ) from the right. If WANTZ = .FALSE., then Z is unreferenced.

LDZ Input parameter.

LDZ is INTEGER

LDA is the leading dimension of Z just as declared in the calling procedure. LDZ.GE.N.

V Output parameter.

V is DOUBLE PRECISION

V is an array, dimension (LDV,NSHFTS/2) .

LDV Input parameter.

LDV is INTEGER

LDV is the leading dimension of V as declared in the calling procedure. LDV.GE.3.

U Output parameter.

U is DOUBLE PRECISION

U is an array, dimension (LDU,3*NSHFTS-3) .

LDU Input parameter.

LDU is INTEGER

LDU is the leading dimension of U just as declared in the in the calling subroutine. LDU.GE.3*NSHFTS-3.

NH Input parameter.

NH is INTEGER

NH is the number of columns in array WH available for workspace. NH.GE.1.

WH Output parameter.

WH is DOUBLE PRECISION

WH is an array, dimension (LDWH, NH) .

LDWH Input parameter.

LDWH is INTEGER

Leading dimension of WH just as declared in the calling procedure. LDWH.GE.3*NSHFTS-3.

NV Input parameter.

NV is INTEGER

NV is the number of rows in WV agailable for workspace. NV.GE.1.

WV Output parameter.

WV is DOUBLE PRECISION

WV is an array, dimension (LDWV,3*NSHFTS-3) .

LDWV Input parameter.

LDWV is INTEGER

LDWV is the leading dimension of WV as declared in the in the calling subroutine. LDWV.GE.NV.

Related Information

For this routine in other precisions, please see *claqr5*, *slaqr5* and *zlaqr5*.

4.17.291 dlaqsb

dlaqsb equilibrates a symmetric band matrix A using the scaling factors in the vector S.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlaqsb(UPLO, N, KD, AB, LDAB, S, SCOND, AMAX, EQUED)
```

C specification:

```
#include "armpl.h"

void dlaqsb_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
             double *ab, const armpl_int_t *ldab, const double *s,
             const double *scond, const double *amax, char *equed, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored. = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of super-diagonals of the matrix A if UPLO = 'U', or the number of sub-diagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, if INFO = 0, the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T of the band matrix A, in the same storage format as A.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

S Input parameter.

S is DOUBLE PRECISION

S is an array, dimension (N). The scale factors for A.

SCOND Input parameter.

SCOND is DOUBLE PRECISION

Ratio of the smallest S(i) to the largest S(i).

AMAX Input parameter.

AMAX is DOUBLE PRECISION

Absolute value of largest matrix entry.

EQUED Output parameter.

EQUED is CHARACTER*1

Specifies whether or not equilibration was done. = 'N': No equilibration. = 'Y': Equilibration was done, i.e., A has been replaced by diag(S) * A * diag(S).

Related Information

For this routine in other precisions, please see *claqsb*, *slaqsb* and *zlaqsb*.

4.17.292 dlaqsp

dlaqsp equilibrates a symmetric matrix A using the scaling factors in the vector S.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlaqsp(UPLO, N, AP, S, SCOND, AMAX, EQUED)
```

C specification:

```
#include "armpl.h"

void dlaqsp_(const char *uplo, const armpl_int_t *n, double *ap,
             const double *s, const double *scond, const double *amax,
             char *equed, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored. = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is DOUBLE PRECISION

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

On exit, the equilibrated matrix: diag(S) * A * diag(S), in the same storage format as A.

S Input parameter.

S is DOUBLE PRECISION

S is an array, dimension (N). The scale factors for A.

SCOND Input parameter.

SCOND is DOUBLE PRECISION

Ratio of the smallest S(i) to the largest S(i).

AMAX Input parameter.

AMAX is DOUBLE PRECISION

Absolute value of largest matrix entry.

EQUED Output parameter.

EQUED is CHARACTER*1

Specifies whether or not equilibration was done. = 'N': No equilibration. = 'Y': Equilibration was done, i.e., A has been replaced by diag(S) * A * diag(S).

Related Information

For this routine in other precisions, please see *claqsp*, *slaqsp* and *zlaqsp*.

4.17.293 dlaqsy

dlaqsy equilibrates a symmetric matrix A using the scaling factors in the vector S.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlaqsy(UPLO, N, A, LDA, S, SCOND, AMAX, EQUED)
```

C specification:

```
#include "armpl.h"

void dlaqsy_(const char *uplo, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, const double *s, const double *scond,
             const double *amax, char *equed, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored. = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading n by n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n by n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if EQUED = 'Y', the equilibrated matrix: diag(S) * A * diag(S).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(N,1).

S Input parameter.

S is DOUBLE PRECISION

S is an array, dimension (N). The scale factors for A.

SCOND Input parameter.

SCOND is DOUBLE PRECISION

Ratio of the smallest S(i) to the largest S(i).

AMAX Input parameter.

AMAX is DOUBLE PRECISION

Absolute value of largest matrix entry.

EQUED Output parameter.

EQUED is CHARACTER*1

Specifies whether or not equilibration was done. = 'N': No equilibration. = 'Y': Equilibration was done, i.e., A has been replaced by diag(S) * A * diag(S).

Related Information

For this routine in other precisions, please see *claqsy*, *slaqsy* and *zlaqsy*.

4.17.294 dlaqtr

`dlaqtr` solves the real quasi-triangular system

```
op(T)*p = scale*c,                 if LREAL = .TRUE.
```

or the complex quasi-triangular systems

```
op(T + iB)*(p+iq) = scale*(c+id),  if LREAL = .FALSE.
```

in real arithmetic, where T is upper quasi-triangular. If LREAL = .FALSE., then the first diagonal block of T must be 1 by 1, B is the specially structured matrix

```
B = [ b(1) b(2) ... b(n) ]
    [       w            ]
    [           w        ]
    [               .    ]
    [                   w ]
```

op(A) = A or A^T , A^T denotes the transpose of matrix A.

On input, X = [c]. On output, X = [p].

```
[ d ]                  [ q ]
```

This subroutine is designed for the condition number estimation in routine DTRSNA.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlaqtr(LTRAN, LREAL, N, T, LDT, B, W, SCALE, X, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dlaqtr_(const armpl_int_t *ltran, const armpl_int_t *lreal,
             const armpl_int_t *n, const double *t, const armpl_int_t *ldt,
             const double *b, const double *w, double *scale, double *x,
             double *work, armpl_int_t *info);
```

Parameters

LTRAN Input parameter.

LTRAN is LOGICAL

On entry, LTRAN specifies the option of conjugate transpose: = .FALSE., op(T+i*B) = T+i*B, = .TRUE., op(T+i*B) = $(T+i*B)^T$.

LREAL Input parameter.

L‘‘REAL‘‘ is LOGICAL

On entry, L‘‘REAL‘‘ specifies the input matrix structure: = .FALSE., the input is complex = .TRUE., the input is real

N Input parameter.

N is INTEGER

On entry, N specifies the order of T+i*B. N >= 0.

T Input parameter.

T is DOUBLE PRECISION

T is an array, dimension (LDT, N). On entry, T contains a matrix in Schur canonical form. If L‘‘REAL‘‘ = .FALSE., then the first diagonal block of T mu be 1 by 1.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the matrix T. LDT >= max(1, N).

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (N). On entry, B contains the elements to form the matrix B as described above. If L‘‘REAL‘‘ = .TRUE., B is not referenced.

W Input parameter.

W is DOUBLE PRECISION

On entry, W is the diagonal element of the matrix B. If L‘‘REAL‘‘ = .TRUE., W is not referenced.

SCALE Output parameter.

SCALE is DOUBLE PRECISION

On exit, SCALE is the scale factor.

X Input and output parameter.

X is DOUBLE PRECISION

X is an array, dimension (2*N). On entry, X contains the right hand side of the system. On exit, X is overwritten by the solution.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

On exit, INFO is set to 0: successful exit. 1: the some diagonal 1 by 1 block has been perturbed by a small number SMIN to keep nonsingularity. 2: the some diagonal 2 by 2 block has been perturbed by a small number in DLALN2 to keep nonsingularity. NOTE: In the interests of speed, this routine does not check the inputs for errors.

Related Information

For this routine in other precisions, please see *slaqtr*.

4.17.295 dlar1v

dlar1v computes the (scaled) r-th column of the inverse of the sumbmatrix in rows B1 through BN of the tridiagonal matrix L D L^T - sigma I. When sigma is close to an eigenvalue, the computed vector is an accurate eigenvector. Usually, r corresponds to the index where the eigenvector is largest in magnitude. The following steps

accomplish this computation : (a) Stationary qd transform, L D L^T - sigma I = L(+) D(+) $L(+)^T$, (b) Progressive qd transform, L D L^T - sigma I = U(-) D(-) $U(-)^T$, (c) Computation of the diagonal elements of the inverse of

```
L D L**T - sigma I by combining the above transforms, and choosing
r as the index where the diagonal of the inverse is (one of the)
largest in magnitude.
```

(d) Computation of the (scaled) r-th column of the inverse using the

```
twisted factorization obtained by combining the top part of the
the stationary and the bottom part of the progressive transform.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dlar1v(N, B1, BN, LAMBDA, D, L, LD, LLD, PIVMIN, GAPTOL, Z, WANTNC,
                  NEGCNT, ZTZ, MINGMA, R, ISUPPZ, NRMINV, RESID, RQCORR,
                  WORK)
```

C specification:

```
#include "armpl.h"

void dlar1v_(const armpl_int_t *n, const armpl_int_t *b1,
             const armpl_int_t *bn, const double *lambda, const double *d,
             const double *l, const double *ld, const double *lld,
             const double *pivmin, const double *gaptol, double *z,
             const armpl_int_t *wantnc, armpl_int_t *negcnt, double *ztz,
             double *mingma, armpl_int_t *r, armpl_int_t *isuppz,
             double *nrminv, double *resid, double *rqcorr, double *work);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix L D L^T .

B1 Input parameter.

B1 is INTEGER

First index of the submatrix of L D L^T .

BN Input parameter.

BN is INTEGER

Last index of the submatrix of L D L^T .

LAMBDA Input parameter.

LAMBDA is DOUBLE PRECISION

The shift. In order to compute an accurate eigenvector, LAMBDA should be a good approximation to an eigenvalue of L D L^T .

L Input parameter.

L is DOUBLE PRECISION

L is an array, dimension (N-1). The (n-1) subdiagonal elements of the unit bidiagonal matrix L, in elements 1 to N-1.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The n diagonal elements of the diagonal matrix D.

LD Input parameter.

LD is DOUBLE PRECISION

LD is an array, dimension (N-1). The n-1 elements L(i)*D(i).

LLD Input parameter.

LLD is DOUBLE PRECISION

LLD is an array, dimension (N-1). The n-1 elements L(i)*L(i)*D(i).

PIVMIN Input parameter.

PIVMIN is DOUBLE PRECISION

The minimum pivot in the Sturm sequence.

GAPTOL Input parameter.

GAPTOL is DOUBLE PRECISION

Tolerance that indicates when eigenvector entries are negligible w.r.t. their contribution to the residual.

Z Input and output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (N). On input, all entries of Z must be set to 0. On output, Z contains the (scaled) r-th column of the inverse. The scaling is such that Z(R) equals 1.

WANTNC Input parameter.

WANTNC is LOGICAL

Specifies whether NEGCNT has to be computed.

NEGCNT Output parameter.

NEGCNT is INTEGER

If WANTNC is .TRUE. then NEGCNT = the number of pivots < pivmin in the matrix factorization L D L^T , and NEGCNT = -1 otherwise.

ZTZ Output parameter.

ZTZ is DOUBLE PRECISION

The square of the 2-norm of Z.

MINGMA Output parameter.

MINGMA is DOUBLE PRECISION

The reciprocal of the largest (in magnitude) diagonal element of the inverse of L D L^T - sigma I.

R Input and output parameter.

R is INTEGER

The twist index for the twisted factorization used to compute Z. On input, 0 <= R <= N. If R is input as 0, R is set to the index where (L D L^T - sigma I)^{-1} is largest in magnitude. If 1 <= R <= N, R is unchanged. On output, R contains the twist index used to compute Z. Ideally, R designates the position of the maximum entry in the eigenvector.

ISUPPZ Output parameter.

ISUPPZ is INTEGER array, dimension (2)

The support of the vector in Z, i.e., the vector Z is nonzero only in elements ISUPPZ(1) through ISUPPZ(2).

NRMINV Output parameter.

NRMINV is DOUBLE PRECISION

NRMINV = 1/SQRT(ZTZ)

RESID Output parameter.

RESID is DOUBLE PRECISION

The residual of the FP vector. RESID = ABS(MINGMA)/SQRT(ZTZ)

RQCORR Output parameter.

RQCORR is DOUBLE PRECISION

The Rayleigh Quotient correction to LAMBDA. RQCORR = MINGMA*TMP

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (4*N) .

Related Information

For this routine in other precisions, please see *clar1v*, *slar1v* and *zlar1v*.

4.17.296 dlar2v

dlar2v applies a vector of real plane rotations from both sides to a sequence of 2-by-2 real symmetric matrices, defined by the elements of the vectors x, y and z. For i = 1,2,...,n

```
( x(i)  z(i) ) := (  c(i)  s(i) ) ( x(i)  z(i) ) ( c(i) -s(i) )
( z(i)  y(i) )    ( -s(i)  c(i) ) ( z(i)  y(i) ) ( s(i)  c(i) )
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dlar2v(N, X, Y, Z, INCX, C, S, INCC)
```

C specification:

```
#include "armpl.h"

void dlar2v_(const armpl_int_t *n, double *x, double *y, double *z,
             const armpl_int_t *incx, const double *c, const double *s,
             const armpl_int_t *incc);
```

Parameters

N Input parameter.

N is INTEGER

The number of plane rotations to be applied.

X Input and output parameter.

X is DOUBLE PRECISION

X is an array,. dimension (1+(N-1)*INCX) The vector x.

Y Input and output parameter.

Y is DOUBLE PRECISION

Y is an array,. dimension (1+(N-1)*INCX) The vector y.

Z Input and output parameter.

Z is DOUBLE PRECISION

Z is an array,. dimension (1+(N-1)*INCX) The vector z.

INCX Input parameter.

INCX is INTEGER

The increment between elements of X, Y and Z. INCX > 0.

C Input parameter.

C is DOUBLE PRECISION

C is an array, dimension (1+(N-1)*INCC). The cosines of the plane rotations.

S Input parameter.

S is DOUBLE PRECISION

S is an array, dimension (1+(N-1)*INCC). The sines of the plane rotations.

INCC Input parameter.

INCC is INTEGER

The increment between elements of C and S. INCC > 0.

Related Information

For this routine in other precisions, please see *clar2v*, *slar2v* and *zlar2v*.

4.17.297 dlarf

dlarf applies a real elementary reflector H to a real m by n matrix C, from either the left or the right. H is represented in the form

```
H = I - tau * v * v**T
```

where tau is a real scalar and v is a real vector.

If tau = 0, then H is taken to be the unit matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlarf(SIDE, M, N, V, INCV, TAU, C, LDC, WORK)
```

C specification:

```
#include "armpl.h"

void dlarf_(const char *side, const armpl_int_t *m, const armpl_int_t *n,
            const double *v, const armpl_int_t *incv, const double *tau,
            double *c, const armpl_int_t *ldc, double *work, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': form H * C = 'R': form C * H

M Input parameter.

M is INTEGER

The number of rows of the matrix C.

N Input parameter.

N is INTEGER

The number of columns of the matrix C.

V Input parameter.

V is DOUBLE PRECISION

V is an array, dimension. (1 + (M-1)*abs(INCV)) if SIDE = 'L' or (1 + (N-1)*abs(INCV)) if SIDE = 'R' The vector v in the representation of H. V is not used if TAU = 0.

INCV Input parameter.

INCV is INTEGER

The increment between elements of v. INCV <> 0.

TAU Input parameter.

TAU is DOUBLE PRECISION

The value tau in the representation of H.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, N). On entry, the m by n matrix C. On exit, C is overwritten by the matrix H * C if SIDE = 'L', or C * H if SIDE = 'R'.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension. (N) if SIDE = 'L' or (M) if SIDE = 'R'

Related Information

For this routine in other precisions, please see *clarf*, *slarf* and *zlarf*.

4.17.298 dlarfb

dlarfb applies a real block reflector H or its transpose H^T to a real m by n matrix C, from either the left or the right.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlarfb(SIDE, TRANS, DIRECT, STOREV, M, N, K, V, LDV, T, LDT, C,
                  LDC, WORK, LDWORK)
```

C specification:

```
#include "armpl.h"

void dlarfb_(const char *side, const char *trans, const char *direct,
             const char *storev, const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *k, const double *v, const armpl_int_t *ldv,
             const double *t, const armpl_int_t *ldt, double *c,
             const armpl_int_t *ldc, double *work, const armpl_int_t *ldwork,
             ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply H or H^T from the Left = 'R': apply H or H^T from the Right

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply H (No transpose) = 'T': apply H^T (Transpose)

DIRECT Input parameter.

DIRECT is CHARACTER*1

Indicates how H is formed from a product of elementary reflectors = 'F': H = H(1) H(2) . . . H(k) (Forward) = 'B': H = H(k) . . . H(2) H(1) (Backward)

STOREV Input parameter.

STOREV is CHARACTER*1

Indicates how the vectors which define the elementary reflectors are stored: = 'C': Columnwise = 'R': Rowwise

M Input parameter.

M is INTEGER

The number of rows of the matrix C.

N Input parameter.

N is INTEGER

The number of columns of the matrix C.

K Input parameter.

K is INTEGER

The order of the matrix T (= the number of elementary reflectors whose product defines the block reflector).

V Input parameter.

V is DOUBLE PRECISION

V is an array, dimension. (LDV, K) if STOREV = 'C' (LDV, M) if STOREV = 'R' and SIDE = 'L' (LDV, N) if STOREV = 'R' and SIDE = 'R' The matrix V. See Further Details.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. If STOREV = 'C' and SIDE = 'L', LDV >= max(1, M); if STOREV = 'C' and SIDE = 'R', LDV >= max(1, N); if STOREV = 'R', LDV >= K.

T Input parameter.

T is DOUBLE PRECISION

T is an array, dimension (LDT, K). The triangular k by k matrix T in the representation of the block reflector.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= K.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, N). On entry, the m by n matrix C. On exit, C is overwritten by H* C or H^T * C or C*H or C*H^T .

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (LDWORK, K) .

LDWORK Input parameter.

LDWORK is INTEGER

The leading dimension of the array WORK. If SIDE = 'L', LDWORK >= max(1, N); if SIDE = 'R', LDWORK >= max(1, M).

Related Information

For this routine in other precisions, please see *clarfb*, *slarfb* and *zlarfb*. It also exists with a native C interface as *LAPACKE_dlarfb*.

4.17.299 dlarfg

`dlarfg` generates a real elementary reflector H of order n, such that

```
H * ( alpha ) = ( beta ),   H**T * H = I.
    (   x   )   (   0  )
```

where alpha and beta are scalars, and x is an (n-1)-element real vector. H is represented in the form

```
H = I - tau * ( 1 ) * ( 1 v**T ) ,
              ( v )
```

where tau is a real scalar and v is a real (n-1)-element vector.

If the elements of x are all zero, then tau = 0 and H is taken to be the unit matrix.

Otherwise 1 <= tau <= 2.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlarfg(N, ALPHA, X, INCX, TAU)
```

C specification:

```
#include "armpl.h"

void dlarfg_(const armpl_int_t *n, double *alpha, double *x,
             const armpl_int_t *incx, double *tau);
```

Parameters

N Input parameter.

N is INTEGER

The order of the elementary reflector.

ALPHA Input and output parameter.

ALPHA is DOUBLE PRECISION

On entry, the value alpha. On exit, it is overwritten with the value beta.

X Input and output parameter.

X is DOUBLE PRECISION

X is an array, dimension. (1+(N-2)*abs(INCX)) On entry, the vector x. On exit, it is overwritten with the vector v.

INCX Input parameter.

INCX is INTEGER

The increment between elements of X. INCX > 0.

TAU Output parameter.

TAU is DOUBLE PRECISION

The value tau.

Related Information

For this routine in other precisions, please see *clarfg*, *slarfg* and *zlarfg*. It also exists with a native C interface as *LAPACKE_dlarfg*.

4.17.300 dlarfgp

dlarfgp generates a real elementary reflector H of order n, such that

```
H * ( alpha ) = ( beta ),   H**T * H = I.
    (   x   )   (   0  )
```

where alpha and beta are scalars, beta is non-negative, and x is an (n-1)-element real vector. H is represented in the form

```
H = I - tau * ( 1 ) * ( 1 v**T ) ,
              ( v )
```

where tau is a real scalar and v is a real (n-1)-element vector.

If the elements of x are all zero, then tau = 0 and H is taken to be the unit matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlarfgp(N, ALPHA, X, INCX, TAU)
```

C specification:

```
#include "armpl.h"

void dlarfgp_(const armpl_int_t *n, double *alpha, double *x,
              const armpl_int_t *incx, double *tau);
```

Parameters

N Input parameter.

N is INTEGER

The order of the elementary reflector.

ALPHA Input and output parameter.

ALPHA is DOUBLE PRECISION

On entry, the value alpha. On exit, it is overwritten with the value beta.

X Input and output parameter.

X is DOUBLE PRECISION

X is an array, dimension. (1+(N-2)*abs(INCX)) On entry, the vector x. On exit, it is overwritten with the vector v.

INCX Input parameter.

INCX is INTEGER

The increment between elements of X. INCX > 0.

TAU Output parameter.

TAU is DOUBLE PRECISION

The value tau.

Related Information

For this routine in other precisions, please see *clarfgp*, *slarfgp* and *zlarfgp*.

4.17.301 dlarft

dlarft forms the triangular factor T of a real block reflector H of order n, which is defined as a product of k elementary reflectors.

If DIRECT = 'F', H = H(1) H(2) . . . H(k) and T is upper triangular;

If DIRECT = 'B', H = H(k) . . . H(2) H(1) and T is lower triangular.

If STOREV = 'C', the vector which defines the elementary reflector H(i) is stored in the i-th column of the array V, and

```
H  =  I - V * T * V**T
```

If STOREV = 'R', the vector which defines the elementary reflector H(i) is stored in the i-th row of the array V, and

```
H  =  I - V**T * T * V
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dlarft(DIRECT, STOREV, N, K, V, LDV, TAU, T, LDT)
```

C specification:

```
#include "armpl.h"

void dlarft_(const char *direct, const char *storev, const armpl_int_t *n,
             const armpl_int_t *k, const double *v, const armpl_int_t *ldv,
             const double *tau, double *t, const armpl_int_t *ldt, ... );
```

Parameters

DIRECT Input parameter.

DIRECT is CHARACTER*1

Specifies the order in which the elementary reflectors are multiplied to form the block reflector: = 'F': H = H(1) H(2) . . . H(k) (Forward) = 'B': H = H(k) . . . H(2) H(1) (Backward)

STOREV Input parameter.

STOREV is CHARACTER*1

Specifies how the vectors which define the elementary reflectors are stored (see also Further Details): = 'C': columnwise = 'R': rowwise

N Input parameter.

N is INTEGER

The order of the block reflector H. N >= 0.

K Input parameter.

K is INTEGER

The order of the triangular factor T (= the number of elementary reflectors). K >= 1.

V Input parameter.

V is DOUBLE PRECISION

V is an array, dimension. (LDV, K) if STOREV = 'C' (LDV, N) if STOREV = 'R' The matrix V. See further details.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. If STOREV = 'C', LDV >= max(1, N); if STOREV = 'R', LDV >= K.

TAU Input parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i).

T Output parameter.

T is DOUBLE PRECISION

T is an array, dimension (LDT, K). The k by k triangular factor T of the block reflector. If DIRECT = 'F', T is upper triangular; if DIRECT = 'B', T is lower triangular. The rest of the array is not used.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= K.

Related Information

For this routine in other precisions, please see *clarft*, *slarft* and *zlarft*. It also exists with a native C interface as *LAPACKE_dlarft*.

4.17.302 dlarfx

dlarfx applies a real elementary reflector H to a real m by n matrix C, from either the left or the right. H is represented in the form

```
H = I - tau * v * v**T
```

where tau is a real scalar and v is a real vector.

If tau = 0, then H is taken to be the unit matrix

This version uses inline code if H has order < 11.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlarfx(SIDE, M, N, V, TAU, C, LDC, WORK)
```

C specification:

```
#include "armpl.h"

void dlarfx_(const char *side, const armpl_int_t *m, const armpl_int_t *n,
             const double *v, const double *tau, double *c,
             const armpl_int_t *ldc, double *work, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': form H * C = 'R': form C * H

M Input parameter.

M is INTEGER

The number of rows of the matrix C.

N Input parameter.

N is INTEGER

The number of columns of the matrix C.

V Input parameter.

V is DOUBLE PRECISION

V is an array, dimension (M) if SIDE = 'L'. or (N) if SIDE = 'R' The vector v in the representation of H.

TAU Input parameter.

TAU is DOUBLE PRECISION

The value tau in the representation of H.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, N). On entry, the m by n matrix C. On exit, C is overwritten by the matrix H * C if SIDE = 'L', or C * H if SIDE = 'R'.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDA >= (1, M).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension. (N) if SIDE = 'L' or (M) if SIDE = 'R' WORK is not referenced if H has order < 11.

Related Information

For this routine in other precisions, please see *clarfx*, *slarfx* and *zlarfx*. It also exists with a native C interface as *LAPACKE_dlarfx*.

4.17.303 dlarfy

`dlarfy` applies an elementary reflector, or Householder matrix, H, to an n x n symmetric matrix C, from both the left and the right.

H is represented in the form

```
H = I - tau * v * v'
```

where tau is a scalar and v is a vector.

If tau is zero, then H is taken to be the unit matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlarfy(UPLO, N, V, INCV, TAU, C, LDC, WORK)
```

C specification:

```
#include "armpl.h"

void dlarfy_(const char *uplo, const armpl_int_t *n, const double *v,
             const armpl_int_t *incv, const double *tau, double *c,
             const armpl_int_t *ldc, double *work, ... );
```

Parameters

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

Specifies whether the upper or lower triangular part of the symmetric matrix `C` is stored. = '`U`': Upper triangle = '`L`': Lower triangle

N Input parameter.

`N` is `INTEGER`

The number of rows and columns of the matrix `C`. `N` >= 0.

V Input parameter.

`V` is `DOUBLE PRECISION`

`V` is an array, dimension. (1 + (N-1)*abs(`INCV`)) The vector v as described above.

INCV Input parameter.

`INCV` is `INTEGER`

The increment between successive elements of v. `INCV` must not be zero.

TAU Input parameter.

TAU is DOUBLE PRECISION

The value tau as described above.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, N). On entry, the matrix C. On exit, C is overwritten by H * C * H'.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (N) .

Related Information

For this routine in other precisions, please see *clarfy*, *slarfy* and *zlarfy*.

4.17.304 dlargv

dlargv generates a vector of real plane rotations, determined by elements of the real vectors x and y. For i = 1,2,...,n

```
(  c(i)  s(i) ) ( x(i) ) = ( a(i) )
( -s(i)  c(i) ) ( y(i) ) = (   0  )
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dlargv(N, X, INCX, Y, INCY, C, INCC)
```

C specification:

```
#include "armpl.h"

void dlargv_(const armpl_int_t *n, double *x, const armpl_int_t *incx,
             double *y, const armpl_int_t *incy, double *c,
             const armpl_int_t *incc);
```

Parameters

N Input parameter.

N is INTEGER

The number of plane rotations to be generated.

X Input and output parameter.

X is DOUBLE PRECISION

X is an array,. dimension (1+(N-1)*INCX) On entry, the vector x. On exit, x(i) is overwritten by a(i), for i = 1,. . . ,n.

INCX Input parameter.

INCX is INTEGER

The increment between elements of X. INCX > 0.

Y Input and output parameter.

Y is DOUBLE PRECISION

Y is an array,. dimension (1+(N-1)*INCY) On entry, the vector y. On exit, the sines of the plane rotations.

INCY Input parameter.

INCY is INTEGER

The increment between elements of Y. INCY > 0.

C Output parameter.

C is DOUBLE PRECISION

C is an array, dimension (1+(N-1)*INCC). The cosines of the plane rotations.

INCC Input parameter.

INCC is INTEGER

The increment between elements of C. INCC > 0.

Related Information

For this routine in other precisions, please see *clargv*, *slargv* and *zlargv*.

4.17.305 dlarnv

dlarnv returns a vector of n random real numbers from a uniform or normal distribution.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlarnv(IDIST, ISEED, N, X)
```

C specification:

```
#include "armpl.h"

void dlarnv_(const armpl_int_t *idist, armpl_int_t *iseed,
             const armpl_int_t *n, double *x);
```

Parameters

IDIST Input parameter.

IDIST is INTEGER

Specifies the distribution of the random numbers: = 1: uniform (0,1) = 2: uniform (-1,1) = 3: normal (0,1)

ISEED Input and output parameter.

ISEED is INTEGER array, dimension (4)

On entry, the seed of the random number generator; the array elements must be between 0 and 4095, and ISEED(4) must be odd. On exit, the seed is updated.

N Input parameter.

N is INTEGER

The number of random numbers to be generated.

X Output parameter.

X is DOUBLE PRECISION

X is an array, dimension (N). The generated random numbers.

Related Information

For this routine in other precisions, please see *clarnv*, *slarnv* and *zlarnv*. It also exists with a native C interface as *LAPACKE_dlarnv*.

4.17.306 dlarra

Compute the splitting points with threshold SPLTOL. dlarra sets any "small" off-diagonal elements to zero.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlarra(N, D, E, E2, SPLTOL, TNRM, NSPLIT, ISPLIT, INFO)
```

C specification:

```
#include "armpl.h"

void dlarra_(const armpl_int_t *n, const double *d, double *e, double *e2,
             const double *spltol, const double *tnrm, armpl_int_t *nsplit,
             armpl_int_t *isplit, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix. N > 0.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry, the N diagonal elements of the tridiagonal matrix T.

E Input and output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N). On entry, the first (N-1) entries contain the subdiagonal elements of the tridiagonal matrix T; E(N) need not be set. On exit, the entries E(ISPLIT(I)), 1 <= I <= NSPLIT, are set to zero, the other entries of E are untouched.

E2 Input and output parameter.

E2 is DOUBLE PRECISION

E2 is an array, dimension (N). On entry, the first (N-1) entries contain the SQUARES of the subdiagonal elements of the tridiagonal matrix T; E2(N) need not be set. On exit, the entries E2(ISPLIT(I)), 1 <= I <= NSPLIT, have been set to zero

SPLTOL Input parameter.

SPLTOL is DOUBLE PRECISION

The threshold for splitting. Two criteria can be used: SPLTOL<0 : criterion based on absolute off-diagonal value SPLTOL>0 : criterion that preserves relative accuracy

TNRM Input parameter.

TNRM is DOUBLE PRECISION

The norm of the matrix.

NSPLIT Output parameter.

NSPLIT is INTEGER

The number of blocks T splits into. 1 <= NSPLIT <= N.

ISPLIT Output parameter.

ISPLIT is INTEGER array, dimension (N)

The splitting points, at which T breaks up into blocks. The first block consists of rows/columns 1 to ISPLIT(1), the second of rows/columns ISPLIT(1)+1 through ISPLIT(2), etc., and the NSPLIT-th consists of rows/columns ISPLIT(NSPLIT-1)+1 through ISPLIT(NSPLIT)=N.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit

Related Information

For this routine in other precisions, please see *slarra*.

4.17.307 dlarrb

Given the relatively robust representation(RRR) L D L^T, dlarrb does "limited" bisection to refine the eigenvalues of L D L^T, W(IFIRST-OFFSET) through W(ILAST-OFFSET), to more accuracy. Initial guesses for these eigenvalues are input in W, the corresponding estimate of the error in these guesses and their gaps are input in WERR and WGAP, respectively. During bisection, intervals [left, right] are maintained by storing their mid-points and semi-widths in the arrays W and WERR respectively.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlarrb(N, D, LLD, IFIRST, ILAST, RTOL1, RTOL2, OFFSET, W, WGAP,
                  WERR, WORK, IWORK, PIVMIN, SPDIAM, TWIST, INFO)
```

C specification:

```
#include "armpl.h"

void dlarrb_(const armpl_int_t *n, const double *d, const double *lld,
             const armpl_int_t *ifirst, const armpl_int_t *ilast,
             const double *rtol1, const double *rtol2,
             const armpl_int_t *offset, double *w, double *wgap, double *werr,
             double *work, armpl_int_t *iwork, const double *pivmin,
             const double *spdiam, const armpl_int_t *twist,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The N diagonal elements of the diagonal matrix D.

LLD Input parameter.

LLD is DOUBLE PRECISION

LLD is an array, dimension (N-1). The (N-1) elements L(i)*L(i)*D(i).

IFIRST Input parameter.

IFIRST is INTEGER

The index of the first eigenvalue to be computed.

ILAST Input parameter.

ILAST is INTEGER

The index of the last eigenvalue to be computed.

RTOL1 Input parameter.

RTOL1 is DOUBLE PRECISION

RTOL2 Input parameter.

RTOL2 is DOUBLE PRECISION

Tolerance for the convergence of the bisection intervals. An interval [LEFT,RIGHT] has converged if RIGHT-LEFT.LT.MAX(RTOL1*GAP, RTOL2*MAX(|LEFT|,|RIGHT|)) where GAP is the (estimated) distance to the nearest eigenvalue.

OFFSET Input parameter.

OFFSET is INTEGER

Offset for the arrays W, WGAP and WERR, i.e., the IFIRST-OFFSET through ILAST-OFFSET elements of these arrays are to be used.

W Input and output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). On input, W(IFIRST-OFFSET) through W(ILAST-OFFSET) are estimates of the eigenvalues of L D L^T indexed IFIRST through ILAST. On output, these estimates are refined.

WGAP Input and output parameter.

WGAP is DOUBLE PRECISION

WGAP is an array, dimension (N-1). On input, the (estimated) gaps between consecutive eigenvalues of L D L^T, i.e., WGAP(I-OFFSET) is the gap between eigenvalues I and I+1. Note that if IFIRST.EQ.ILAST then WGAP(IFIRST-OFFSET) must be set to ZERO. On output, these gaps are refined.

WERR Input and output parameter.

WERR is DOUBLE PRECISION

WERR is an array, dimension (N). On input, WERR(IFIRST-OFFSET) through WERR(ILAST-OFFSET) are the errors in the estimates of the corresponding elements in W. On output, these errors are refined.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (2*N). Workspace.

IWORK Output parameter.

IWORK is INTEGER array, dimension (2*N)

Workspace.

PIVMIN Input parameter.

PIVMIN is DOUBLE PRECISION

The minimum pivot in the Sturm sequence.

SPDIAM Input parameter.

SPDIAM is DOUBLE PRECISION

The spectral diameter of the matrix.

TWIST Input parameter.

TWIST is INTEGER

The twist index for the twisted factorization that is used for the negcount. TWIST = N: Compute negcount from L D L^T - LAMBDA I = L+ D+ L+^T TWIST = 1: Compute negcount from L D L^T - LAMBDA I = U- D- U-^T TWIST = R: Compute negcount from L D L^T - LAMBDA I = N(r) D(r) N(r)

INFO Output parameter.

INFO is INTEGER

Error flag.

Related Information

For this routine in other precisions, please see *slarrb*.

4.17.308 dlarrc

Find the number of eigenvalues of the symmetric tridiagonal matrix T that are in the interval (VL,VU] if JOBT = 'T', and of L D L^T if JOBT = 'L'.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlarrc(JOBT, N, VL, VU, D, E, PIVMIN, EIGCNT, LCNT, RCNT, INFO)
```

C specification:

```
#include "armpl.h"

void dlarrc_(const char *jobt, const armpl_int_t *n, const double *vl,
             const double *vu, const double *d, const double *e,
             const double *pivmin, armpl_int_t *eigcnt, armpl_int_t *lcnt,
             armpl_int_t *rcnt, armpl_int_t *info, ... );
```

Parameters

JOBT Input parameter.

JOBT is CHARACTER*1

= 'T': Compute Sturm count for matrix T. = 'L': Compute Sturm count for matrix L D L^T.

N Input parameter.

N is INTEGER

The order of the matrix. N > 0.

VL Input parameter.

VL is DOUBLE PRECISION

The lower bound for the eigenvalues.

VU Input parameter.

VU is DOUBLE PRECISION

The upper bound for the eigenvalues.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). JOBT = 'T': The N diagonal elements of the tridiagonal matrix T. JOBT = 'L': The N diagonal elements of the diagonal matrix D.

E Input parameter.

E is DOUBLE PRECISION

E is an array, dimension (N). JOBT = 'T': The N-1 offdiagonal elements of the matrix T. JOBT = 'L': The N-1 offdiagonal elements of the matrix L.

PIVMIN Input parameter.

PIVMIN is DOUBLE PRECISION

The minimum pivot in the Sturm sequence for T.

EIGCNT Output parameter.

EIGCNT is INTEGER

The number of eigenvalues of the symmetric tridiagonal matrix T that are in the interval (VL,VU]

LCNT Output parameter.

LCNT is INTEGER

RCNT Output parameter.

RCNT is INTEGER

The left and right negcounts of the interval.

INFO Output parameter.

INFO is INTEGER

Related Information

For this routine in other precisions, please see *slarrc*.

4.17.309 dlarrd

dlarrd computes the eigenvalues of a symmetric tridiagonal matrix T to suitable accuracy. This is an auxiliary code to be called from DSTEMR. The user may ask for all eigenvalues, all eigenvalues in the half-open interval (VL, VU], or the IL-th through IU-th eigenvalues.

To avoid overflow, the matrix must be scaled so that its largest element is no greater than overflow**(1/2) * underflow**(1/4) in absolute value, and for greatest accuracy, it should not be much smaller than that.

See W. Kahan "Accurate Eigenvalues of a Symmetric Tridiagonal Matrix", Report CS41, Computer Science Dept., Stanford University, July 21, 1966.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlarrd(RANGE, ORDER, N, VL, VU, IL, IU, GERS, RELTOL, D, E, E2,
                  PIVMIN, NSPLIT, ISPLIT, M, W, WERR, WL, WU, IBLOCK, INDEXW,
                  WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dlarrd_(const char *range, const char *order, const armpl_int_t *n,
             const double *vl, const double *vu, const armpl_int_t *il,
             const armpl_int_t *iu, const double *gers, const double *reltol,
             const double *d, const double *e, const double *e2,
             const double *pivmin, const armpl_int_t *nsplit,
             const armpl_int_t *isplit, armpl_int_t *m, double *w,
             double *werr, double *wl, double *wu, armpl_int_t *iblock,
             armpl_int_t *indexw, double *work, armpl_int_t *iwork,
             armpl_int_t *info, ... );
```

Parameters

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': ("All") all eigenvalues will be found. = 'V': ("Value") all eigenvalues in the half-open interval (VL, VU] will be found. = 'I': ("Index") the IL-th through IU-th eigenvalues (of the entire matrix) will be found.

ORDER Input parameter.

ORDER is CHARACTER*1

= 'B': ("By Block") the eigenvalues will be grouped by split-off block (see IBLOCK, ISPLIT) and ordered from smallest to largest within the block. = 'E': ("Entire matrix") the eigenvalues for the entire matrix will be ordered from smallest to largest.

N Input parameter.

N is INTEGER

The order of the tridiagonal matrix T. N >= 0.

VL Input parameter.

VL is DOUBLE PRECISION

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. Eigenvalues less than or equal to VL, or greater than VU, will not be returned. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is DOUBLE PRECISION

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. Eigenvalues less than or equal to VL, or greater than VU, will not be returned. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

GERS Input parameter.

GERS is DOUBLE PRECISION

GERS is an array, dimension (2*N). The N Gerschgorin intervals (the i-th Gerschgorin interval is (GERS(2*i-1), GERS(2*i)).

RELTOL Input parameter.

RELTOL is DOUBLE PRECISION

The minimum relative width of an interval. When an interval is narrower than RELTOL times the larger (in magnitude) endpoint, then it is considered to be sufficiently small, i.e., converged. Note: this should always be at least radix*machine epsilon.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The n diagonal elements of the tridiagonal matrix T.

E Input parameter.

E is DOUBLE PRECISION

E is an array, dimension (N-1). The (n-1) off-diagonal elements of the tridiagonal matrix T.

E2 Input parameter.

E2 is DOUBLE PRECISION

E2 is an array, dimension (N-1). The (n-1) squared off-diagonal elements of the tridiagonal matrix T.

PIVMIN Input parameter.

PIVMIN is DOUBLE PRECISION

The minimum pivot allowed in the Sturm sequence for T.

NSPLIT Input parameter.

NSPLIT is INTEGER

The number of diagonal blocks in the matrix T. 1 <= NSPLIT <= N.

ISPLIT Input parameter.

ISPLIT is INTEGER array, dimension (N)

The splitting points, at which T breaks up into submatrices. The first submatrix consists of rows/columns 1 to ISPLIT(1), the second of rows/columns ISPLIT(1)+1 through ISPLIT(2), etc., and the NSPLIT-th consists of rows/columns ISPLIT(NSPLIT-1)+1 through ISPLIT(NSPLIT)=N. (Only the first NSPLIT elements will actually be used, but since the user cannot know a priori what value NSPLIT will have, N words must be reserved for ISPLIT.)

M Output parameter.

M is INTEGER

The actual number of eigenvalues found. 0 <= M <= N. (See also the description of INFO=2,3.)

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). On exit, the first M elements of W will contain the eigenvalue approximations. DLARRD computes an interval I_j = (a_j, b_j] that includes eigenvalue j. The eigenvalue approximation is given as the interval midpoint W(j)= (a_j + b_j)/2. The corresponding error is bounded by WERR(j) = abs(a_j - b_j)/2

WERR Output parameter.

WERR is DOUBLE PRECISION

WERR is an array, dimension (N). The error bound on the corresponding eigenvalue approximation in W.

WL Output parameter.

WL is DOUBLE PRECISION

WU Output parameter.

WU is DOUBLE PRECISION

The interval (WL, WU] contains all the wanted eigenvalues. If RANGE='V', then WL=VL and WU=VU. If RANGE='A', then WL and WU are the global Gerschgorin bounds on the spectrum. If RANGE='I', then WL and WU are computed by DLAEBZ from the index range specified.

IBLOCK Output parameter.

IBLOCK is INTEGER array, dimension (N)

At each row/column j where E(j) is zero or small, the matrix T is considered to split into a block diagonal matrix. On exit, if INFO = 0, IBLOCK(i) specifies to which block (from 1 to the number of blocks) the eigenvalue W(i) belongs. (DLARRD may use the remaining N-M elements as workspace.)

INDEXW Output parameter.

INDEXW is INTEGER array, dimension (N)

The indices of the eigenvalues within each block (submatrix); for example, INDEXW(i)= j and IBLOCK(i)=k imply that the i-th eigenvalue W(i) is the j-th eigenvalue in block k.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (4*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (3*N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: some or all of the eigenvalues failed to converge or were not computed: =1 or 3: Bisection failed to converge for some eigenvalues; these eigenvalues are flagged by a negative block number. The effect is that the eigenvalues may not be as accurate as the absolute and relative tolerances. This is generally caused by unexpectedly inaccurate arithmetic. =2 or 3: RANGE='I' only: Not all of the eigenvalues IL:IU were found. Effect: M < IU+1-IL Cause: non-monotonic arithmetic, causing the Sturm sequence to be non-monotonic. Cure: recalculate, using RANGE='A', and pick out eigenvalues IL:IU. In some cases, increasing the PARAMETER "FUDGE" may make things work. = 4: RANGE='I', and the Gershgorin interval initially used was too small. No eigenvalues were computed. Probable cause: your machine has sloppy floating-point arithmetic. Cure: Increase the PARAMETER "FUDGE", recompile, and try again.

Related Information

For this routine in other precisions, please see *slarrd*.

4.17.310 dlarre

To find the desired eigenvalues of a given real symmetric tridiagonal matrix T, dlarre sets any "small" off-diagonal elements to zero, and for each unreduced block T_i, it finds (a) a suitable shift at one end of the block's spectrum, (b) the base representation, T_i - sigma_i I = L_i D_i L_i^T, and (c) eigenvalues of each L_i D_i L_i^T. The representations and eigenvalues found are then used by DSTEMR to compute the eigenvectors of T. The accuracy varies depending on whether bisection is used to find a few eigenvalues or the dqds algorithm (subroutine DLASQ2) to conpute all and then discard any unwanted one. As an added benefit, dlarre also outputs the n Gerschgorin intervals for the matrices L_i D_i L_i^T.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlarre(RANGE, N, VL, VU, IL, IU, D, E, E2, RTOL1, RTOL2, SPLTOL,
                  NSPLIT, ISPLIT, M, W, WERR, WGAP, IBLOCK, INDEXW, GERS,
                  PIVMIN, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dlarre_(const char *range, const armpl_int_t *n, double *vl, double *vu,
```

(continues on next page)

(continued from previous page)

```
                const armpl_int_t *il, const armpl_int_t *iu, double *d,
                double *e, double *e2, const double *rtol1, const double *rtol2,
                const double *spltol, armpl_int_t *nsplit, armpl_int_t *isplit,
                armpl_int_t *m, double *w, double *werr, double *wgap,
                armpl_int_t *iblock, armpl_int_t *indexw, double *gers,
                double *pivmin, double *work, armpl_int_t *iwork,
                armpl_int_t *info, ... );
```

Parameters

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': ("All") all eigenvalues will be found. = 'V': ("Value") all eigenvalues in the half-open interval (VL, VU] will be found. = 'I': ("Index") the IL-th through IU-th eigenvalues (of the entire matrix) will be found.

N Input parameter.

N is INTEGER

The order of the matrix. N > 0.

VL Input and output parameter.

VL is DOUBLE PRECISION

If RANGE='V', the lower bound for the eigenvalues. Eigenvalues less than or equal to VL, or greater than VU, will not be returned. VL < VU. If RANGE='I' or ='A', DLARRE computes bounds on the desired part of the spectrum.

VU Input and output parameter.

VU is DOUBLE PRECISION

If RANGE='V', the upper bound for the eigenvalues. Eigenvalues less than or equal to VL, or greater than VU, will not be returned. VL < VU. If RANGE='I' or ='A', DLARRE computes bounds on the desired part of the spectrum.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry, the N diagonal elements of the tridiagonal matrix T. On exit, the N diagonal elements of the diagonal matrices D_i.

E Input and output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N). On entry, the first (N-1) entries contain the subdiagonal elements of the tridiagonal matrix T; E(N) need not be set. On exit, E contains the subdiagonal elements of the unit bidiagonal matrices L_i. The entries E(ISPLIT(I)), 1 <= I <= NSPLIT, contain the base points sigma_i on output.

E2 Input and output parameter.

E2 is DOUBLE PRECISION

E2 is an array, dimension (N). On entry, the first (N-1) entries contain the SQUARES of the subdiagonal elements of the tridiagonal matrix T; E2(N) need not be set. On exit, the entries E2(ISPLIT(I)), 1 <= I <= NSPLIT, have been set to zero

RTOL1 Input parameter.

RTOL1 is DOUBLE PRECISION

RTOL2 Input parameter.

RTOL2 is DOUBLE PRECISION

Parameters for bisection. An interval [LEFT,RIGHT] has converged if RIGHT-LEFT.LT.MAX(RTOL1*GAP, RTOL2*MAX(|LEFT|,|RIGHT|))

SPLTOL Input parameter.

SPLTOL is DOUBLE PRECISION

The threshold for splitting.

NSPLIT Output parameter.

NSPLIT is INTEGER

The number of blocks T splits into. 1 <= NSPLIT <= N.

ISPLIT Output parameter.

ISPLIT is INTEGER array, dimension (N)

The splitting points, at which T breaks up into blocks. The first block consists of rows/columns 1 to ISPLIT(1), the second of rows/columns ISPLIT(1)+1 through ISPLIT(2), etc., and the NSPLIT-th consists of rows/columns ISPLIT(NSPLIT-1)+1 through ISPLIT(NSPLIT)=N.

M Output parameter.

M is INTEGER

The total number of eigenvalues (of all L_i D_i L_i^T) found.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). The first M elements contain the eigenvalues. The eigenvalues of each of the blocks, L_i D_i L_i^T, are sorted in ascending order (DLARRE may use the remaining N-M elements as workspace).

WERR Output parameter.

WERR is DOUBLE PRECISION

WERR is an array, dimension (N). The error bound on the corresponding eigenvalue in W.

WGAP Output parameter.

WGAP is DOUBLE PRECISION

WGAP is an array, dimension (N). The separation from the right neighbor eigenvalue in W. The gap is only with respect to the eigenvalues of the same block as each block has its own representation tree. Exception: at the right end of a block we store the left gap

IBLOCK Output parameter.

IBLOCK is INTEGER array, dimension (N)

The indices of the blocks (submatrices) associated with the corresponding eigenvalues in W; IBLOCK(i)=1 if eigenvalue W(i) belongs to the first block from the top, =2 if W(i) belongs to the second block, etc.

INDEXW Output parameter.

INDEXW is INTEGER array, dimension (N)

The indices of the eigenvalues within each block (submatrix); for example, INDEXW(i)= 10 and IBLOCK(i)=2 imply that the i-th eigenvalue W(i) is the 10-th eigenvalue in block 2

GERS Output parameter.

GERS is DOUBLE PRECISION

GERS is an array, dimension (2*N). The N Gerschgorin intervals (the i-th Gerschgorin interval is (GERS(2*i-1), GERS(2*i)).

PIVMIN Output parameter.

PIVMIN is DOUBLE PRECISION

The minimum pivot in the Sturm sequence for T.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (6*N). Workspace.

IWORK Output parameter.

IWORK is INTEGER array, dimension (5*N)

Workspace.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit > 0: A problem occurred in DLARRE. < 0: One of the called subroutines signaled an internal problem. Needs inspection of the corresponding parameter IINFO for further information.

=-1: Problem in DLARRD. = 2: No base representation could be found in MAXTRY iterations. Increasing MAXTRY and recompilation might be a remedy. =-3: Problem in DLARRB when computing the refined root representation for DLASQ2. =-4: Problem in DLARRB when preforming bisection on the desired part of the spectrum. =-5: Problem in DLASQ2. =-6: Problem in DLASQ2.

Related Information

For this routine in other precisions, please see *slarre*.

4.17.311 dlarrf

Given the initial representation L D L^T and its cluster of close eigenvalues (in a relative measure), W(CLSTRT), W(CLSTRT+1), ... W(CLEND), dlarrf finds a new relatively robust representation L D L^T - SIGMA I = L(+) D(+) L(+)^T such that at least one of the eigenvalues of L(+) D(+) L(+)^T is relatively isolated.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlarrf(N, D, L, LD, CLSTRT, CLEND, W, WGAP, WERR, SPDIAM, CLGAPL,
                  CLGAPR, PIVMIN, SIGMA, DPLUS, LPLUS, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dlarrf_(const armpl_int_t *n, const double *d, const double *l,
             const double *ld, const armpl_int_t *clstrt,
             const armpl_int_t *clend, const double *w, double *wgap,
             const double *werr, const double *spdiam, const double *clgapl,
             double *clgapr, const double *pivmin, double *sigma,
             double *dplus, double *lplus, double *work, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix (subblock, if the matrix split).

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The N diagonal elements of the diagonal matrix D.

L Input parameter.

L is DOUBLE PRECISION

L is an array, dimension (N-1). The (N-1) subdiagonal elements of the unit bidiagonal matrix L.

LD Input parameter.

LD is DOUBLE PRECISION

LD is an array, dimension (N-1). The (N-1) elements L(i)*D(i).

CLSTRT Input parameter.

CLSTRT is INTEGER

The index of the first eigenvalue in the cluster.

CLEND Input parameter.

CLEND is INTEGER

The index of the last eigenvalue in the cluster.

W Input parameter.

W is DOUBLE PRECISION

W is an array, dimension. dimension is >= (CLEND-CLSTRT+1) The eigenvalue APPROXIMATIONS of L D L^T in ascending order. W(CLSTRT) through W(CLEND) form the cluster of relatively close eigenalues.

WGAP Input and output parameter.

WGAP is DOUBLE PRECISION

WGAP is an array, dimension. dimension is >= (CLEND-CLSTRT+1) The separation from the right neighbor eigenvalue in W.

WERR Input parameter.

WERR is DOUBLE PRECISION

WERR is an array, dimension. dimension is >= (CLEND-CLSTRT+1) WERR contain the semiwidth of the uncertainty interval of the corresponding eigenvalue APPROXIMATION in W

SPDIAM Input parameter.

SPDIAM is DOUBLE PRECISION

estimate of the spectral diameter obtained from the Gerschgorin intervals

CLGAPL Input parameter.

CLGAPL is DOUBLE PRECISION

CLGAPR Input parameter.

CLGAPR is DOUBLE PRECISION

absolute gap on each end of the cluster. Set by the calling routine to protect against shifts too close to eigenvalues outside the cluster.

PIVMIN Input parameter.

PIVMIN is DOUBLE PRECISION

The minimum pivot allowed in the Sturm sequence.

SIGMA Output parameter.

SIGMA is DOUBLE PRECISION

The shift used to form L(+) D(+) L(+)^T.

DPLUS Output parameter.

DPLUS is DOUBLE PRECISION

DPLUS is an array, dimension (N). The N diagonal elements of the diagonal matrix D(+).

LPLUS Output parameter.

LPLUS is DOUBLE PRECISION

LPLUS is an array, dimension (N-1). The first (N-1) elements of LPLUS contain the subdiagonal elements of the unit bidiagonal matrix L(+).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (2*N). Workspace.

INFO Output parameter.

INFO is INTEGER

Signals processing OK (=0) or failure (=1)

Related Information

For this routine in other precisions, please see *slarrf*.

4.17.312 dlarrj

Given the initial eigenvalue approximations of T, dlarrj does bisection to refine the eigenvalues of T, W(IFIRST-OFFSET) through W(ILAST-OFFSET), to more accuracy. Initial guesses for these eigenvalues are input in W, the corresponding estimate of the error in these guesses in WERR. During bisection, intervals [left, right] are maintained by storing their mid-points and semi-widths in the arrays W and WERR respectively.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlarrj(N, D, E2, IFIRST, ILAST, RTOL, OFFSET, W, WERR, WORK, IWORK,
                  PIVMIN, SPDIAM, INFO)
```

C specification:

```
#include "armpl.h"

void dlarrj_(const armpl_int_t *n, const double *d, const double *e2,
             const armpl_int_t *ifirst, const armpl_int_t *ilast,
             const double *rtol, const armpl_int_t *offset, double *w,
             double *werr, double *work, armpl_int_t *iwork,
             const double *pivmin, const double *spdiam, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The N diagonal elements of T.

E2 Input parameter.

E2 is DOUBLE PRECISION

E2 is an array, dimension (N-1). The Squares of the (N-1) subdiagonal elements of T.

IFIRST Input parameter.

IFIRST is INTEGER

The index of the first eigenvalue to be computed.

ILAST Input parameter.

ILAST is INTEGER

The index of the last eigenvalue to be computed.

RTOL Input parameter.

RTOL is DOUBLE PRECISION

Tolerance for the convergence of the bisection intervals. An interval [LEFT,RIGHT] has converged if RIGHT-LEFT.LT.RTOL*MAX(|LEFT|,|RIGHT|).

OFFSET Input parameter.

OFFSET is INTEGER

Offset for the arrays W and WERR, i.e., the IFIRST-OFFSET through ILAST-OFFSET elements of these arrays are to be used.

W Input and output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). On input, W(IFIRST-OFFSET) through W(ILAST-OFFSET) are estimates of the eigenvalues of L D L^T indexed IFIRST through ILAST. On output, these estimates are refined.

WERR Input and output parameter.

WERR is DOUBLE PRECISION

WERR is an array, dimension (N). On input, WERR(IFIRST-OFFSET) through WERR(ILAST-OFFSET) are the errors in the estimates of the corresponding elements in W. On output, these errors are refined.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (2*N). Workspace.

IWORK Output parameter.

IWORK is INTEGER array, dimension (2*N)

Workspace.

PIVMIN Input parameter.

PIVMIN is DOUBLE PRECISION

The minimum pivot in the Sturm sequence for T.

SPDIAM Input parameter.

SPDIAM is DOUBLE PRECISION

The spectral diameter of T.

INFO Output parameter.

INFO is INTEGER

Error flag.

Related Information

For this routine in other precisions, please see *slarrj*.

4.17.313 dlarrk

dlarrk computes one eigenvalue of a symmetric tridiagonal matrix T to suitable accuracy. This is an auxiliary code to be called from DSTEMR.

To avoid overflow, the matrix must be scaled so that its largest element is no greater than overflow**(1/2) * underflow**(1/4) in absolute value, and for greatest accuracy, it should not be much smaller than that.

See W. Kahan “Accurate Eigenvalues of a Symmetric Tridiagonal Matrix”, Report CS41, Computer Science Dept., Stanford University, July 21, 1966.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlarrk(N, IW, GL, GU, D, E2, PIVMIN, RELTOL, W, WERR, INFO)
```

C specification:

```
#include "armpl.h"

void dlarrk_(const armpl_int_t *n, const armpl_int_t *iw, const double *gl,
             const double *gu, const double *d, const double *e2,
             const double *pivmin, const double *reltol, double *w,
             double *werr, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the tridiagonal matrix T. N >= 0.

IW Input parameter.

IW is INTEGER

The index of the eigenvalues to be returned.

GL Input parameter.

GL is DOUBLE PRECISION

GU Input parameter.

GU is DOUBLE PRECISION

An upper and a lower bound on the eigenvalue.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The n diagonal elements of the tridiagonal matrix T.

E2 Input parameter.

E2 is DOUBLE PRECISION

E2 is an array, dimension (N-1). The (n-1) squared off-diagonal elements of the tridiagonal matrix T.

PIVMIN Input parameter.

PIVMIN is DOUBLE PRECISION

The minimum pivot allowed in the Sturm sequence for T.

RELTOL Input parameter.

RELTOL is DOUBLE PRECISION

The minimum relative width of an interval. When an interval is narrower than RELTOL times the larger (in magnitude) endpoint, then it is considered to be sufficiently small, i.e., converged. Note: this should always be at least radix*machine epsilon.

W Output parameter.

W is DOUBLE PRECISION

WERR Output parameter.

WERR is DOUBLE PRECISION

The error bound on the corresponding eigenvalue approximation in W.

INFO Output parameter.

INFO is INTEGER

= 0: Eigenvalue converged = -1: Eigenvalue did NOT converge

Related Information

For this routine in other precisions, please see *slarrk*.

4.17.314 dlarrr

Perform tests to decide whether the symmetric tridiagonal matrix T warrants expensive computations which guarantee high relative accuracy in the eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlarrr(N, D, E, INFO)
```

C specification:

```
#include "armpl.h"

void dlarrr_(const armpl_int_t *n, const double *d, double *e,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix. N > 0.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The N diagonal elements of the tridiagonal matrix T.

E Input and output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N). On entry, the first (N-1) entries contain the subdiagonal elements of the tridiagonal matrix T; E(N) is set to ZERO.

INFO Output parameter.

INFO is INTEGER

INFO = 0(default) : the matrix warrants computations preserving relative accuracy. INFO = 1 : the matrix warrants computations guaranteeing only absolute accuracy.

Related Information

For this routine in other precisions, please see *slarrr*.

4.17.315 dlarrv

dlarrv computes the eigenvectors of the tridiagonal matrix T = L D L^T given L, D and APPROXIMATIONS to the eigenvalues of L D L^T . The input eigenvalues should have been computed by DLARRE.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlarrv(N, VL, VU, D, L, PIVMIN, ISPLIT, M, DOL, DOU, MINRGP, RTOL1,
                  RTOL2, W, WERR, WGAP, IBLOCK, INDEXW, GERS, Z, LDZ, ISUPPZ,
                  WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dlarrv_(const armpl_int_t *n, const double *vl, const double *vu,
             double *d, double *l, double *pivmin, const armpl_int_t *isplit,
             const armpl_int_t *m, const armpl_int_t *dol,
             const armpl_int_t *dou, const double *minrgp,
             const double *rtol1, const double *rtol2, double *w,
             double *werr, double *wgap, const armpl_int_t *iblock,
             const armpl_int_t *indexw, const double *gers, double *z,
             const armpl_int_t *ldz, armpl_int_t *isuppz, double *work,
             armpl_int_t *iwork, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix. N >= 0.

VL Input parameter.

VL is DOUBLE PRECISION

Lower bound of the interval that contains the desired eigenvalues. VL < VU. Needed to compute gaps on the left or right end of the extremal eigenvalues in the desired RANGE.

VU Input parameter.

VU is DOUBLE PRECISION

Upper bound of the interval that contains the desired eigenvalues. VL < VU. Note: VU is currently not used by this implementation of DLARRV, VU is passed to DLARRV because it could be used compute gaps on the right end of the extremal eigenvalues. However, with not much initial accuracy in LAMBDA and VU, the formula can lead to an overestimation of the right gap and thus to inadequately early RQI 'convergence'. This is currently prevented this by forcing a small right gap. And so it turns out that VU is currently not used by this implementation of DLARRV.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry, the N diagonal elements of the diagonal matrix D. On exit, D may be overwritten.

L Input and output parameter.

L is DOUBLE PRECISION

L is an array, dimension (N). On entry, the (N-1) subdiagonal elements of the unit bidiagonal matrix L are in elements 1 to N-1 of L (if the matrix is not split.) At the end of each block is stored the corresponding shift as given by DLARRE. On exit, L is overwritten.

PIVMIN Input parameter.

PIVMIN is DOUBLE PRECISION

The minimum pivot allowed in the Sturm sequence.

ISPLIT Input parameter.

ISPLIT is INTEGER array, dimension (N)

The splitting points, at which T breaks up into blocks. The first block consists of rows/columns 1 to ISPLIT(1), the second of rows/columns ISPLIT(1)+1 through ISPLIT(2), etc.

M Input parameter.

M is INTEGER

The total number of input eigenvalues. 0 <= M <= N.

DOL Input parameter.

DOL is INTEGER

DOU Input parameter.

DOU is INTEGER

If the user wants to compute only selected eigenvectors from all the eigenvalues supplied, he can specify an index range DOL:DOU. Or else the setting DOL=1, DOU=M should be applied. Note that DOL and DOU refer to the order in which the eigenvalues are stored in W. If the user wants to compute only selected eigenpairs, then the columns DOL-1 to DOU+1 of the eigenvector space Z contain the computed eigenvectors. All other columns of Z are set to zero.

MINRGP Input parameter.

MINRGP is DOUBLE PRECISION

RTOL1 Input parameter.

RTOL1 is DOUBLE PRECISION

RTOL2 Input parameter.

RTOL2 is DOUBLE PRECISION

Parameters for bisection. An interval [LEFT,RIGHT] has converged if RIGHT-LEFT.LT.MAX(RTOL1*GAP, RTOL2*MAX(|LEFT|,|RIGHT|))

W Input and output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). The first M elements of W contain the APPROXIMATE eigenvalues for which eigenvectors are to be computed. The eigenvalues should be grouped by split-off block and ordered from smallest to largest within the block (The output array W from DLARRE is expected here). Furthermore, they are with respect to the shift of the corresponding root representation for their block. On exit, W holds the eigenvalues of the UNshifted matrix.

WERR Input and output parameter.

WERR is DOUBLE PRECISION

WERR is an array, dimension (N). The first M elements contain the semiwidth of the uncertainty interval of the corresponding eigenvalue in W

WGAP Input and output parameter.

WGAP is DOUBLE PRECISION

WGAP is an array, dimension (N). The separation from the right neighbor eigenvalue in W.

IBLOCK Input parameter.

IBLOCK is INTEGER array, dimension (N)

The indices of the blocks (submatrices) associated with the corresponding eigenvalues in W; IBLOCK(i)=1 if eigenvalue W(i) belongs to the first block from the top, =2 if W(i) belongs to the second block, etc.

INDEXW Input parameter.

INDEXW is INTEGER array, dimension (N)

The indices of the eigenvalues within each block (submatrix); for example, INDEXW(i)= 10 and IBLOCK(i)=2 imply that the i-th eigenvalue W(i) is the 10-th eigenvalue in the second block.

GERS Input parameter.

GERS is DOUBLE PRECISION

GERS is an array, dimension (2*N). The N Gerschgorin intervals (the i-th Gerschgorin interval is (GERS(2*i-1), GERS(2*i)). The Gerschgorin intervals should be computed from the original UNshifted matrix.

Z Output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, max(1, M)). If INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix T corresponding to the input eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). Note: the user must ensure that at least max(1, M) columns are supplied in the array Z.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

ISUPPZ Output parameter.

ISUPPZ is INTEGER array, dimension (2*max(1, M))

The support of the eigenvectors in Z, i.e., the indices indicating the nonzero elements in Z. The I-th eigenvector is nonzero only in elements ISUPPZ(2*I-1) through ISUPPZ(2*I).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (12*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (7*N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit

> 0: A problem occurred in DLARRV. < 0: One of the called subroutines signaled an internal problem. Needs inspection of the corresponding parameter IINFO for further information.

=-1: Problem in DLARRB when refining a child's eigenvalues. =-2: Problem in DLARRF when computing the RRR of a child. When a child is inside a tight cluster, it can be difficult to find an RRR. A partial remedy from the user's point of view is to make the parameter MINRGP smaller and recompile. However, as the orthogonality of the computed vectors is proportional to 1/MINRGP, the user should be aware that he might be trading in precision when he decreases MINRGP. =-3: Problem in DLARRB when refining a single eigenvalue after the Rayleigh correction was rejected. = 5: The Rayleigh Quotient Iteration failed to converge to full accuracy in MAXITR steps.

Related Information

For this routine in other precisions, please see *clarrv*, *slarrv* and *zlarrv*.

4.17.316 dlarscl2

DLARSCL2 performs a reciprocal diagonal scaling on an vector:

```
x <-- inv(D) * x
```

where the diagonal matrix D is stored as a vector.

Eventually to be replaced by BLAS_dge_diag_scale in the new BLAS standard.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlarscl2(M, N, D, X, LDX)
```

C specification:

```
#include "armpl.h"

void dlarscl2_(const armpl_int_t *m, const armpl_int_t *n, const double *d,
               double *x, const armpl_int_t *ldx);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of D and X. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of X. N >= 0.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (M). Diagonal matrix D, stored as a vector of length M.

X Input and output parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, N). On entry, the vector X to be scaled by D. On exit, the scaled vector.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the vector X. LDX >= M.

Related Information

For this routine in other precisions, please see *clarscl2*, *slarscl2* and *zlarscl2*.

4.17.317 dlartg

`dlartg` generate a plane rotation so that

```
[  CS  SN  ]  .  [ F ]  =  [ R ]   where CS**2 + SN**2 = 1.
[ -SN  CS  ]     [ G ]     [ 0 ]
```

This is a slower, more accurate version of the BLAS1 routine DROTG, with the following other differences:

```
 F and G are unchanged on return.
If G=0, then CS=1 and SN=0.
If F=0 and (G .ne. 0), then CS=0 and SN=1 without doing any
   floating point operations (saves work in DBDSQR when
   there are zeros on the diagonal).
```

If F exceeds G in magnitude, CS will be positive.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlartg(F, G, CS, SN, R)
```

C specification:

```
#include "armpl.h"

void dlartg_(const double *f, const double *g, double *cs, double *sn,
             double *r);
```

Parameters

F Input parameter.

F is DOUBLE PRECISION

The first component of vector to be rotated.

G Input parameter.

G is DOUBLE PRECISION

The second component of vector to be rotated.

CS Output parameter.

CS is DOUBLE PRECISION

The cosine of the rotation.

SN Output parameter.

SN is DOUBLE PRECISION

The sine of the rotation.

R Output parameter.

R is DOUBLE PRECISION

The nonzero component of the rotated vector.

This version has a few statements commented out for thread safety (machine parameters are computed on each entry). 10 feb 03, SJH.

Related Information

For this routine in other precisions, please see *clartg*, *slartg* and *zlartg*.

4.17.318 dlartgp

`dlartgp` generates a plane rotation so that

```
[  CS  SN  ]  .  [ F ]  =  [ R ]   where CS**2 + SN**2 = 1.
[ -SN  CS  ]     [ G ]     [ 0 ]
```

This is a slower, more accurate version of the Level 1 BLAS routine DROTG, with the following other differences:

```
F and G are unchanged on return.
If G=0, then CS=(+/-)1 and SN=0.
If F=0 and (G .ne. 0), then CS=0 and SN=(+/-)1.
```

The sign is chosen so that R >= 0.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlartgp(F, G, CS, SN, R)
```

C specification:

```
#include "armpl.h"

void dlartgp_(const double *f, const double *g, double *cs, double *sn,
              double *r);
```

Parameters

F Input parameter.

F is DOUBLE PRECISION

The first component of vector to be rotated.

G Input parameter.

G is DOUBLE PRECISION

The second component of vector to be rotated.

CS Output parameter.

CS is DOUBLE PRECISION

The cosine of the rotation.

SN Output parameter.

SN is DOUBLE PRECISION

The sine of the rotation.

R Output parameter.

R is DOUBLE PRECISION

The nonzero component of the rotated vector.

This version has a few statements commented out for thread safety (machine parameters are computed on each entry). 10 feb 03, SJH.

Related Information

For this routine in other precisions, please see *slartgp*. It also exists with a native C interface as *LAPACKE_dlartgp*.

4.17.319 dlartgs

dlartgs generates a plane rotation designed to introduce a bulge in Golub-Reinsch-style implicit QR iteration for the bidiagonal SVD problem. X and Y are the top-row entries, and SIGMA is the shift. The computed CS and SN define a plane rotation satisfying

```
[  CS  SN  ]  .  [ X^2 - SIGMA ]  =  [ R ],
[ -SN  CS  ]     [    X * Y    ]     [ 0 ]
```

with R nonnegative. If X^2 - SIGMA and X * Y are 0, then the rotation is by PI/2.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlartgs(X, Y, SIGMA, CS, SN)
```

C specification:

```
#include "armpl.h"

void dlartgs_(const double *x, const double *y, const double *sigma,
              double *cs, double *sn);
```

Parameters

X Input parameter.

X is DOUBLE PRECISION

The (1,1) entry of an upper bidiagonal matrix.

Y Input parameter.

Y is DOUBLE PRECISION

The (1,2) entry of an upper bidiagonal matrix.

SIGMA Input parameter.

SIGMA is DOUBLE PRECISION

The shift.

CS Output parameter.

CS is DOUBLE PRECISION

The cosine of the rotation.

SN Output parameter.

SN is DOUBLE PRECISION

The sine of the rotation.

Related Information

For this routine in other precisions, please see *slartgs*. It also exists with a native C interface as *LAPACKE_dlartgs*.

4.17.320 dlartv

dlartv applies a vector of real plane rotations to elements of the real vectors x and y. For i = 1,2,...,n

```
( x(i) ) := (  c(i)  s(i) ) ( x(i) )
( y(i) )    ( -s(i)  c(i) ) ( y(i) )
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dlartv(N, X, INCX, Y, INCY, C, S, INCC)
```

C specification:

```
#include "armpl.h"

void dlartv_(const armpl_int_t *n, double *x, const armpl_int_t *incx,
             double *y, const armpl_int_t *incy, const double *c,
             const double *s, const armpl_int_t *incc);
```

Parameters

N Input parameter.

N is INTEGER

The number of plane rotations to be applied.

X Input and output parameter.

X is DOUBLE PRECISION

X is an array,. dimension (1+(N-1)*INCX) The vector x.

INCX Input parameter.

INCX is INTEGER

The increment between elements of X. INCX > 0.

Y Input and output parameter.

Y is DOUBLE PRECISION

Y is an array,. dimension (1+(N-1)*INCY) The vector y.

INCY Input parameter.

INCY is INTEGER

The increment between elements of Y. INCY > 0.

C Input parameter.

C is DOUBLE PRECISION

C is an array, dimension (1+(N-1)*INCC). The cosines of the plane rotations.

S Input parameter.

S is DOUBLE PRECISION

S is an array, dimension (1+(N-1)*INCC). The sines of the plane rotations.

INCC Input parameter.

INCC is INTEGER

The increment between elements of C and S. INCC > 0.

Related Information

For this routine in other precisions, please see *clartv*, *slartv* and *zlartv*.

4.17.321 dlaruv

dlaruv returns a vector of n random real numbers from a uniform (0,1) distribution (n <= 128).

This is an auxiliary routine called by DLARNV and ZLARNV.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlaruv(ISEED, N, X)
```

C specification:

```
#include "armpl.h"

void dlaruv_(armpl_int_t *iseed, const armpl_int_t *n, double *x);
```

Parameters

ISEED Input and output parameter.

ISEED is INTEGER array, dimension (4)

On entry, the seed of the random number generator; the array elements must be between 0 and 4095, and ISEED(4) must be odd. On exit, the seed is updated.

N Input parameter.

N is INTEGER

The number of random numbers to be generated. N <= 128.

X Output parameter.

X is DOUBLE PRECISION

X is an array, dimension (N). The generated random numbers.

Related Information

For this routine in other precisions, please see *slaruv*.

4.17.322 dlarz

dlarz applies a real elementary reflector H to a real M-by-N matrix C, from either the left or the right. H is represented in the form

```
H = I - tau * v * v**T
```

where tau is a real scalar and v is a real vector.

If tau = 0, then H is taken to be the unit matrix.

H is a product of k elementary reflectors as returned by DTZRZF.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlarz(SIDE, M, N, L, V, INCV, TAU, C, LDC, WORK)
```

C specification:

```
#include "armpl.h"

void dlarz_(const char *side, const armpl_int_t *m, const armpl_int_t *n,
            const armpl_int_t *l, const double *v, const armpl_int_t *incv,
            const double *tau, double *c, const armpl_int_t *ldc,
            double *work, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': form H * C = 'R': form C * H

M Input parameter.

M is INTEGER

The number of rows of the matrix C.

N Input parameter.

N is INTEGER

The number of columns of the matrix C.

L Input parameter.

L is INTEGER

The number of entries of the vector V containing the meaningful part of the Householder vectors. If SIDE = 'L', M >= L >= 0, if SIDE = 'R', N >= L >= 0.

V Input parameter.

V is DOUBLE PRECISION

V is an array, dimension (1+(L-1)*abs(INCV)). The vector v in the representation of H as returned by DTZRZF. V is not used if TAU = 0.

INCV Input parameter.

INCV is INTEGER

The increment between elements of v. INCV <> 0.

TAU Input parameter.

TAU is DOUBLE PRECISION

The value tau in the representation of H.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by the matrix H * C if SIDE = 'L', or C * H if SIDE = 'R'.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension. (N) if SIDE = 'L' or (M) if SIDE = 'R'

Related Information

For this routine in other precisions, please see *clarz*, *slarz* and *zlarz*.

4.17.323 dlarzb

dlarzb applies a real block reflector H or its transpose H^T to a real distributed M-by-N C from the left or the right.

Currently, only STOREV = 'R' and DIRECT = 'B' are supported.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlarzb(SIDE, TRANS, DIRECT, STOREV, M, N, K, L, V, LDV, T, LDT, C,
                  LDC, WORK, LDWORK)
```

C specification:

```
#include "armpl.h"

void dlarzb_(const char *side, const char *trans, const char *direct,
             const char *storev, const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *k, const armpl_int_t *l, const double *v,
             const armpl_int_t *ldv, const double *t, const armpl_int_t *ldt,
             double *c, const armpl_int_t *ldc, double *work,
             const armpl_int_t *ldwork, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply H or H^T from the Left = 'R': apply H or H^T from the Right

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply H (No transpose) = 'C': apply H^T (Transpose)

DIRECT Input parameter.

DIRECT is CHARACTER*1

Indicates how H is formed from a product of elementary reflectors = 'F': H = H(1) H(2) . . . H(k) (Forward, not supported yet) = 'B': H = H(k) . . . H(2) H(1) (Backward)

STOREV Input parameter.

STOREV is CHARACTER*1

Indicates how the vectors which define the elementary reflectors are stored: = 'C': Columnwise (not supported yet) = 'R': Rowwise

M Input parameter.

M is INTEGER

The number of rows of the matrix C.

N Input parameter.

N is INTEGER

The number of columns of the matrix C.

K Input parameter.

K is INTEGER

The order of the matrix T (= the number of elementary reflectors whose product defines the block reflector).

L Input parameter.

L is INTEGER

The number of columns of the matrix V containing the meaningful part of the Householder reflectors. If SIDE = 'L', M >= L >= 0, if SIDE = 'R', N >= L >= 0.

V Input parameter.

V is DOUBLE PRECISION

V is an array, dimension (LDV,NV).. If STOREV = 'C', NV = K; if STOREV = 'R', NV = L.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. If STOREV = 'C', LDV >= L; if STOREV = 'R', LDV >= K.

T Input parameter.

T is DOUBLE PRECISION

T is an array, dimension (LDT, K). The triangular K-by-K matrix T in the representation of the block reflector.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= K.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by H* C or H^T * C or C*H or C*H^T .

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (LDWORK, K) .

LDWORK Input parameter.

LDWORK is INTEGER

The leading dimension of the array WORK. If SIDE = 'L', LDWORK >= max(1, N); if SIDE = 'R', LDWORK >= max(1, M).

Related Information

For this routine in other precisions, please see *clarzb*, *slarzb* and *zlarzb*.

4.17.324 dlarzt

dlarzt forms the triangular factor T of a real block reflector H of order > n, which is defined as a product of k elementary reflectors.

If DIRECT = 'F', H = H(1) H(2) . . . H(k) and T is upper triangular;

If DIRECT = 'B', H = H(k) . . . H(2) H(1) and T is lower triangular.

If STOREV = 'C', the vector which defines the elementary reflector H(i) is stored in the i-th column of the array V, and

```
H  =  I - V * T * V**T
```

If STOREV = 'R', the vector which defines the elementary reflector H(i) is stored in the i-th row of the array V, and

```
H  =  I - V**T * T * V
```

Currently, only STOREV = 'R' and DIRECT = 'B' are supported.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlarzt(DIRECT, STOREV, N, K, V, LDV, TAU, T, LDT)
```

C specification:

```
#include "armpl.h"

void dlarzt_(const char *direct, const char *storev, const armpl_int_t *n,
             const armpl_int_t *k, double *v, const armpl_int_t *ldv,
             const double *tau, double *t, const armpl_int_t *ldt, ... );
```

Parameters

DIRECT Input parameter.

DIRECT is CHARACTER*1

Specifies the order in which the elementary reflectors are multiplied to form the block reflector: = 'F': H = H(1) H(2) . . . H(k) (Forward, not supported yet) = 'B': H = H(k) . . . H(2) H(1) (Backward)

STOREV Input parameter.

STOREV is CHARACTER*1

Specifies how the vectors which define the elementary reflectors are stored (see also Further Details): = 'C': columnwise (not supported yet) = 'R': rowwise

N Input parameter.

N is INTEGER

The order of the block reflector H. N >= 0.

K Input parameter.

K is INTEGER

The order of the triangular factor T (= the number of elementary reflectors). K >= 1.

V Input and output parameter.

V is DOUBLE PRECISION

V is an array, dimension. (LDV, K) if STOREV = 'C' (LDV, N) if STOREV = 'R' The matrix V. See further details.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. If STOREV = 'C', LDV >= max(1, N); if STOREV = 'R', LDV >= K.

TAU Input parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i).

T Output parameter.

T is DOUBLE PRECISION

T is an array, dimension (LDT, K). The k by k triangular factor T of the block reflector. If DIRECT = 'F', T is upper triangular; if DIRECT = 'B', T is lower triangular. The rest of the array is not used.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= K.

Related Information

For this routine in other precisions, please see *clarzt*, *slarzt* and *zlarzt*.

4.17.325 dlas2

dlas2 computes the singular values of the 2-by-2 matrix

```
[  F   G  ]
[  0   H  ].
```

On return, SSMIN is the smaller singular value and SSMAX is the larger singular value.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlas2(F, G, H, SSMIN, SSMAX)
```

C specification:

```
#include "armpl.h"

void dlas2_(const double *f, const double *g, const double *h, double *ssmin,
            double *ssmax);
```

Parameters

F Input parameter.

F is DOUBLE PRECISION

The (1,1) element of the 2-by-2 matrix.

G Input parameter.

G is DOUBLE PRECISION

The (1,2) element of the 2-by-2 matrix.

H Input parameter.

H is DOUBLE PRECISION

The (2,2) element of the 2-by-2 matrix.

SSMIN Output parameter.

SSMIN is DOUBLE PRECISION

The smaller singular value.

SSMAX Output parameter.

SSMAX is DOUBLE PRECISION

The larger singular value.

Related Information

For this routine in other precisions, please see *slas2*.

4.17.326 dlascl

dlascl multiplies the M by N real matrix A by the real scalar CTO/CFROM. This is done without over/underflow as long as the final result CTO*A(I,J)/CFROM does not over/underflow. TYPE specifies that A may be full, upper triangular, lower triangular, upper Hessenberg, or banded.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlascl(TYPE, KL, KU, CFROM, CTO, M, N, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void dlascl_(const char *type, const armpl_int_t *kl, const armpl_int_t *ku,
             const double *cfrom, const double *cto, const armpl_int_t *m,
             const armpl_int_t *n, double *a, const armpl_int_t *lda,
             armpl_int_t *info, ... );
```

Parameters

TYPE Input parameter.

TYPE is CHARACTER*1

TYPE indices the storage type of the input matrix. = 'G': A is a full matrix. = 'L': A is a lower triangular matrix. = 'U': A is an upper triangular matrix. = 'H': A is an upper Hessenberg matrix. = 'B': A is a symmetric band matrix with lower bandwidth KL and upper bandwidth KU and with the only the lower half stored. = 'Q': A is a symmetric band matrix with lower bandwidth KL and upper bandwidth KU and with the only the upper half stored. = 'Z': A is a band matrix with lower bandwidth KL and upper bandwidth KU. See DGBTRF for storage details.

KL Input parameter.

KL is INTEGER

The lower bandwidth of A. Referenced only if TYPE = 'B', 'Q' or 'Z'.

KU Input parameter.

KU is INTEGER

The upper bandwidth of A. Referenced only if TYPE = 'B', 'Q' or 'Z'.

CFROM Input parameter.

CFROM is DOUBLE PRECISION

CTO Input parameter.

CTO is DOUBLE PRECISION

The matrix A is multiplied by CTO/CFROM. A(I,J) is computed without over/underflow if the final result CTO*A(I,J)/CFROM can be represented without over/underflow. CFROM must be nonzero.

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). The matrix to be multiplied by CTO/CFROM. See TYPE for the storage type.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If TYPE = 'G', 'L', 'U', 'H', LDA >= max(1, M); TYPE = 'B', LDA >= KL+1; TYPE = 'Q', LDA >= KU+1; TYPE = 'Z', LDA >= 2*KL+KU+1.

INFO Output parameter.

INFO is INTEGER

0 - successful exit <0 - if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *clascl*, *slascl* and *zlascl*. It also exists with a native C interface as *LAPACKE_dlascl*.

4.17.327 dlascl2

dlascl2 performs a diagonal scaling on a vector:

```
x <-- D * x
```

where the diagonal matrix D is stored as a vector.

Eventually to be replaced by BLAS_dge_diag_scale in the new BLAS standard.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlascl2(M, N, D, X, LDX)
```

C specification:

```
#include "armpl.h"

void dlascl2_(const armpl_int_t *m, const armpl_int_t *n, const double *d,
              double *x, const armpl_int_t *ldx);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of D and X. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of X. N >= 0.

D Input parameter.

D is DOUBLE PRECISION

D is an array, length M. Diagonal matrix D, stored as a vector of length M.

X Input and output parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX, N). On entry, the vector X to be scaled by D. On exit, the scaled vector.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the vector X. LDX >= M.

Related Information

For this routine in other precisions, please see *clascl2*, *slascl2* and *zlascl2*.

4.17.328 dlasd0

Using a divide and conquer approach, dlasd0 computes the singular value decomposition (SVD) of a real upper bidiagonal N-by-M matrix B with diagonal D and offdiagonal E, where M = N + SQRE. The algorithm computes orthogonal matrices U and VT such that B = U * S * VT. The singular values S are overwritten on D.

A related subroutine, DLASDA, computes only the singular values, and optionally, the singular vectors in compact form.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlasd0(N, SQRE, D, E, U, LDU, VT, LDVT, SMLSIZ, IWORK, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dlasd0_(const armpl_int_t *n, const armpl_int_t *sqre, double *d,
             double *e, double *u, const armpl_int_t *ldu, double *vt,
             const armpl_int_t *ldvt, const armpl_int_t *smlsiz,
             armpl_int_t *iwork, double *work, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

On entry, the row dimension of the upper bidiagonal matrix. This is also the dimension of the main diagonal array D.

SQRE Input parameter.

SQRE is INTEGER

Specifies the column dimension of the bidiagonal matrix. = 0: The bidiagonal matrix has column dimension M = N; = 1: The bidiagonal matrix has column dimension M = N+1;

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry D contains the main diagonal of the bidiagonal matrix. On exit D, if INFO = 0, contains its singular values.

E Input and output parameter.

E is DOUBLE PRECISION

E is an array, dimension (M-1). Contains the subdiagonal entries of the bidiagonal matrix. On exit, E has been destroyed.

U Output parameter.

U is DOUBLE PRECISION

U is an array, dimension (LDU, N). On exit, U contains the left singular vectors.

LDU Input parameter.

LDU is INTEGER

On entry, leading dimension of U.

VT Output parameter.

VT is DOUBLE PRECISION

VT is an array, dimension (LDVT, M). On exit, VT^T contains the right singular vectors.

LDVT Input parameter.

LDVT is INTEGER

On entry, leading dimension of VT.

SMLSIZ Input parameter.

SMLSIZ is INTEGER

On entry, maximum size of the subproblems at the bottom of the computation tree.

IWORK Output parameter.

IWORK is INTEGER array, dimension (8*N)

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (3*M2+2*M)** .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = 1, a singular value did not converge

Related Information

For this routine in other precisions, please see *slasd0*.

4.17.329 dlasd1

dlasd1 computes the SVD of an upper bidiagonal N-by-M matrix B, where N = NL + NR + 1 and M = N + SQRE. dlasd1 is called from DLASD0.

A related subroutine DLASD7 handles the case in which the singular values (and the singular vectors in factored form) are desired.

dlasd1 computes the SVD as follows:

```
             ( D1(in)     0     0        0 )
B = U(in) * (   Z1**T     a    Z2**T     b ) * VT(in)
             (   0        0    D2(in)    0 )

  = U(out) * ( D(out) 0) * VT(out)
```

where Z^T = ($Z1^T$ a $Z2^T$ b) = u^T VT^T , and u is a vector of dimension M with ALPHA and BETA in the NL+1 and NL+2 th entries and zeros elsewhere; and the entry b is empty if SQRE = 0.

The left singular vectors of the original matrix are stored in U, and the transpose of the right singular vectors are stored in VT, and the singular values are in D. The algorithm consists of three stages:

```
 The first stage consists of deflating the size of the problem
when there are multiple singular values or when there are zeros in
the Z vector.  For each such occurrence the dimension of the
secular equation problem is reduced by one.  This stage is
performed by the routine DLASD2.

The second stage consists of calculating the updated
singular values. This is done by finding the square roots of the
roots of the secular equation via the routine DLASD4 (as called
by DLASD3). This routine also calculates the singular vectors of
the current problem.

The final stage consists of computing the updated singular vectors
directly using the updated singular values.  The singular vectors
for the current problem are multiplied with the singular vectors
from the overall problem.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dlasd1(NL, NR, SQRE, D, ALPHA, BETA, U, LDU, VT, LDVT, IDXQ, IWORK,
                  WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dlasd1_(const armpl_int_t *nl, const armpl_int_t *nr,
             const armpl_int_t *sqre, double *d, double *alpha, double *beta,
             double *u, const armpl_int_t *ldu, double *vt,
             const armpl_int_t *ldvt, armpl_int_t *idxq, armpl_int_t *iwork,
             double *work, armpl_int_t *info);
```

Parameters

NL Input parameter.

NL is INTEGER

The row dimension of the upper block. NL >= 1.

NR Input parameter.

NR is INTEGER

The row dimension of the lower block. NR >= 1.

SQRE Input parameter.

SQRE is INTEGER

= 0: the lower block is an NR-by-NR square matrix. = 1: the lower block is an NR-by-(NR+1) rectangular matrix.

The bidiagonal matrix has row dimension N = NL + NR + 1, and column dimension M = N + SQRE.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array,. dimension (N = NL+NR+1). On entry D(1:NL,1:NL) contains the singular values of the upper block; and D(NL+2:N) contains the singular values of the lower block. On exit D(1:N) contains the singular values of the modified matrix.

ALPHA Input and output parameter.

ALPHA is DOUBLE PRECISION

Contains the diagonal element associated with the added row.

BETA Input and output parameter.

BETA is DOUBLE PRECISION

Contains the off-diagonal element associated with the added row.

U Input and output parameter.

U is DOUBLE PRECISION

U is an array, dimension(LDU,N). On entry U(1:NL, 1:NL) contains the left singular vectors of the upper block; U(NL+2:N, NL+2:N) contains the left singular vectors of the lower block. On exit U contains the left singular vectors of the bidiagonal matrix.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U. LDU >= max(1, N).

VT Input and output parameter.

VT is DOUBLE PRECISION

VT is an array, dimension(LDVT,M). where M = N + SQRE. On entry VT(1:NL+1, 1:NL+1)^T contains the right singular vectors of the upper block; VT(NL+2:M, NL+2:M)^T contains the right singular vectors of the lower block. On exit VT^T contains the right singular vectors of the bidiagonal matrix.

LDVT Input parameter.

LDVT is INTEGER

The leading dimension of the array VT. LDVT >= max(1, M).

IDXQ Input and output parameter.

IDXQ is INTEGER array, dimension(N)

This contains the permutation which will reintegrate the subproblem just solved back into sorted order, i.e. D(IDXQ(I = 1, N)) will be in ascending order.

IWORK Output parameter.

IWORK is INTEGER array, dimension(4 * N)

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension(3*M2 + 2*M)** .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = 1, a singular value did not converge

Related Information

For this routine in other precisions, please see *slasd1*.

4.17.330 dlasd2

dlasd2 merges the two sets of singular values together into a single sorted set. Then it tries to deflate the size of the problem. There are two ways in which deflation can occur: when two or more singular values are close together or if there is a tiny entry in the Z vector. For each such occurrence the order of the related secular equation problem is reduced by one.

dlasd2 is called from DLASD1.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlasd2(NL, NR, SQRE, K, D, Z, ALPHA, BETA, U, LDU, VT, LDVT,
                  DSIGMA, U2, LDU2, VT2, LDVT2, IDXP, IDX, IDXC, IDXQ, COLTYP,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dlasd2_(const armpl_int_t *nl, const armpl_int_t *nr,
             const armpl_int_t *sqre, armpl_int_t *k, double *d, double *z,
             const double *alpha, const double *beta, double *u,
             const armpl_int_t *ldu, double *vt, const armpl_int_t *ldvt,
             double *dsigma, double *u2, const armpl_int_t *ldu2, double *vt2,
             const armpl_int_t *ldvt2, armpl_int_t *idxp, armpl_int_t *idx,
             armpl_int_t *idxc, armpl_int_t *idxq, armpl_int_t *coltyp,
             armpl_int_t *info);
```

Parameters

NL Input parameter.

NL is INTEGER

The row dimension of the upper block. NL >= 1.

NR Input parameter.

NR is INTEGER

The row dimension of the lower block. NR >= 1.

SQRE Input parameter.

SQRE is INTEGER

= 0: the lower block is an NR-by-NR square matrix. = 1: the lower block is an NR-by-(NR+1) rectangular matrix.

The bidiagonal matrix has N = NL + NR + 1 rows and M = N + SQRE >= N columns.

K Output parameter.

K is INTEGER

Contains the dimension of the non-deflated matrix, This is the order of the related secular equation. 1 <= K <=N.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension(N). On entry D contains the singular values of the two submatrices to be combined. On exit D contains the trailing (N-K) updated singular values (those which were deflated) sorted into increasing order.

Z Output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension(N). On exit Z contains the updating row vector in the secular equation.

ALPHA Input parameter.

ALPHA is DOUBLE PRECISION

Contains the diagonal element associated with the added row.

BETA Input parameter.

BETA is DOUBLE PRECISION

Contains the off-diagonal element associated with the added row.

U Input and output parameter.

U is DOUBLE PRECISION

U is an array, dimension(LDU,N). On entry U contains the left singular vectors of two submatrices in the two square blocks with corners at (1,1), (NL, NL), and (NL+2, NL+2), (N,N). On exit U contains the trailing (N-K) updated left singular vectors (those which were deflated) in its last N-K columns.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U. LDU >= N.

VT Input and output parameter.

VT is DOUBLE PRECISION

VT is an array, dimension(LDVT,M). On entry VT^T contains the right singular vectors of two submatrices in the two square blocks with corners at (1,1), (NL+1, NL+1), and (NL+2, NL+2), (M,M). On exit VT^T contains the trailing (N-K) updated right singular vectors (those which were deflated) in its last N-K columns. In case SQRE =1, the last row of VT spans the right null space.

LDVT Input parameter.

LDVT is INTEGER

The leading dimension of the array VT. LDVT >= M.

DSIGMA Output parameter.

DSIGMA is DOUBLE PRECISION

DSIGMA is an array, dimension (N). Contains a copy of the diagonal elements (K-1 singular values and one zero) in the secular equation.

U2 Output parameter.

U2 is DOUBLE PRECISION

U2 is an array, dimension(LDU2,N). Contains a copy of the first K-1 left singular vectors which will be used by DLASD3 in a matrix multiply (DGEMM) to solve for the new left singular vectors. U2 is arranged into four blocks. The first block contains a column with 1 at NL+1 and zero everywhere else; the second block contains non-zero entries only at and above NL; the third contains non-zero entries only below NL+1; and the fourth is dense.

LDU2 Input parameter.

LDU2 is INTEGER

The leading dimension of the array U2. LDU2 >= N.

VT2 Output parameter.

VT2 is DOUBLE PRECISION

VT2 is an array, dimension(LDVT2,N). $VT2^T$ contains a copy of the first K right singular vectors which will be used by DLASD3 in a matrix multiply (DGEMM) to solve for the new right singular vectors. VT2 is arranged into three blocks. The first block contains a row that corresponds to the special 0 diagonal element in SIGMA; the second block contains non-zeros only at and before NL +1; the third block contains non-zeros only at and after NL +2.

LDVT2 Input parameter.

LDVT2 is INTEGER

The leading dimension of the array VT2. LDVT2 >= M.

IDXP Output parameter.

IDXP is INTEGER array, dimension(N)

This will contain the permutation used to place deflated values of D at the end of the array. On output IDXP(2:K) points to the nondeflated D-values and IDXP(K+1:N) points to the deflated singular values.

IDX Output parameter.

IDX is INTEGER array, dimension(N)

This will contain the permutation used to sort the contents of D into ascending order.

IDXC Output parameter.

IDXC is INTEGER array, dimension(N)

This will contain the permutation used to arrange the columns of the deflated U matrix into three groups: the first group contains non-zero entries only at and above NL, the second contains non-zero entries only below NL+2, and the third is dense.

IDXQ Input and output parameter.

IDXQ is INTEGER array, dimension(N)

This contains the permutation which separately sorts the two sub-problems in D into ascending order. Note that entries in the first hlaf of this permutation must first be moved one position backward; and entries in the second half must first have NL+1 added to their values.

COLTYP Output parameter.

COLTYP is INTEGER array, dimension(N)

As workspace, this will contain a label which will indicate which of the following types a column in the U2 matrix or a row in the VT2 matrix is: 1 : non-zero in the upper half only 2 : non-zero in the lower half only 3 : dense 4 : deflated

On exit, it is an array of dimension 4, with COLTYP(I) being the dimension of the I-th type columns.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *slasd2*.

4.17.331 dlasd3

dlasd3 finds all the square roots of the roots of the secular equation, as defined by the values in D and Z. It makes the appropriate calls to DLASD4 and then updates the singular vectors by matrix multiplication.

This code makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray XMP, Cray YMP, Cray C 90, or Cray 2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

dlasd3 is called from DLASD1.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlasd3(NL, NR, SQRE, K, D, Q, LDQ, DSIGMA, U, LDU, U2, LDU2, VT,
                  LDVT, VT2, LDVT2, IDXC, CTOT, Z, INFO)
```

C specification:

```
#include "armpl.h"

void dlasd3_(const armpl_int_t *nl, const armpl_int_t *nr,
             const armpl_int_t *sqre, const armpl_int_t *k, double *d,
             double *q, const armpl_int_t *ldq, const double *dsigma,
             double *u, const armpl_int_t *ldu, double *u2,
             const armpl_int_t *ldu2, double *vt, const armpl_int_t *ldvt,
             double *vt2, const armpl_int_t *ldvt2, const armpl_int_t *idxc,
             const armpl_int_t *ctot, const double *z, armpl_int_t *info);
```

Parameters

NL Input parameter.

NL is INTEGER

The row dimension of the upper block. NL >= 1.

NR Input parameter.

NR is INTEGER

The row dimension of the lower block. NR >= 1.

SQRE Input parameter.

SQRE is INTEGER

= 0: the lower block is an NR-by-NR square matrix. = 1: the lower block is an NR-by-(NR+1) rectangular matrix.

The bidiagonal matrix has N = NL + NR + 1 rows and M = N + SQRE >= N columns.

K Input parameter.

K is INTEGER

The size of the secular equation, 1 =< K = < N.

D Output parameter.

D is DOUBLE PRECISION

D is an array, dimension(K). On exit the square roots of the roots of the secular equation, in ascending order.

Q Output parameter.

Q is DOUBLE PRECISION

Q is an array, dimension (LDQ, K) .

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= K.

DSIGMA Input and output parameter.

DSIGMA is DOUBLE PRECISION

DSIGMA is an array, dimension(K). The first K elements of this array contain the old roots of the deflated updating problem. These are the poles of the secular equation.

U Output parameter.

U is DOUBLE PRECISION

U is an array, dimension (LDU, N). The last N - K columns of this matrix contain the deflated left singular vectors.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U. LDU >= N.

U2 Input parameter.

U2 is DOUBLE PRECISION

U2 is an array, dimension (LDU2, N). The first K columns of this matrix contain the non-deflated left singular vectors for the split problem.

LDU2 Input parameter.

LDU2 is INTEGER

The leading dimension of the array U2. LDU2 >= N.

VT Output parameter.

VT is DOUBLE PRECISION

VT is an array, dimension (LDVT, M). The last M - K columns of VT^T contain the deflated right singular vectors.

LDVT Input parameter.

LDVT is INTEGER

The leading dimension of the array VT. LDVT >= N.

VT2 Input and output parameter.

VT2 is DOUBLE PRECISION

VT2 is an array, dimension (LDVT2, N). The first K columns of $VT2^T$ contain the non-deflated right singular vectors for the split problem.

LDVT2 Input parameter.

LDVT2 is INTEGER

The leading dimension of the array VT2. LDVT2 >= N.

IDXC Input parameter.

IDXC is INTEGER array, dimension (N)

The permutation used to arrange the columns of U (and rows of VT) into three groups: the first group contains non-zero entries only at and above (or before) NL +1; the second contains non-zero entries only at and below (or after) NL+2; and the third is dense. The first column of U and the row of VT are treated separately, however.

The rows of the singular vectors found by DLASD4 must be likewise permuted before the matrix multiplies can take place.

CTOT Input parameter.

CTOT is INTEGER array, dimension (4)

A count of the total number of the various types of columns in U (or rows in VT), as described in IDXC. The fourth column type is any column which has been deflated.

Z Input and output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (K). The first K elements of this array contain the components of the deflation-adjusted updating row vector.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = 1, a singular value did not converge

Related Information

For this routine in other precisions, please see *slasd3*.

4.17.332 dlasd4

This subroutine computes the square root of the I-th updated eigenvalue of a positive symmetric rank-one modification to a positive diagonal matrix whose entries are given as the squares of the corresponding entries in the array d, and that

```
0 <= D(i) < D(j)  for  i < j
```

and that RHO > 0. This is arranged by the calling routine, and is no loss in generality. The rank-one modified system is thus

```
diag( D ) * diag( D ) +  RHO * Z * Z_transpose.
```

where we assume the Euclidean norm of Z is 1.

The method consists of approximating the rational functions in the secular equation by simpler interpolating rational functions.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlasd4(N, I, D, Z, DELTA, RHO, SIGMA, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dlasd4_(const armpl_int_t *n, const armpl_int_t *i, const double *d,
             const double *z, double *delta, const double *rho, double *sigma,
             double *work, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The length of all arrays.

I Input parameter.

I is INTEGER

The index of the eigenvalue to be computed. 1 <= I <= N.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The original eigenvalues. It is assumed that they are in order, 0 <= D(I) < D(J) for I < J.

Z Input parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (N). The components of the updating vector.

DELTA Output parameter.

DELTA is DOUBLE PRECISION

DELTA is an array, dimension (N). If N .ne. 1, DELTA contains (D(j) - sigma_I) in its j-th component. If N = 1, then DELTA(1) = 1. The vector DELTA contains the information necessary to construct the (singular) eigenvectors.

RHO Input parameter.

RHO is DOUBLE PRECISION

The scalar in the symmetric updating formula.

SIGMA Output parameter.

SIGMA is DOUBLE PRECISION

The computed sigma_I, the I-th updated eigenvalue.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (N). If N .ne. 1, WORK contains (D(j) + sigma_I) in its j-th component. If N = 1, then WORK(1) = 1.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit > 0: if INFO = 1, the updating process failed.

Related Information

For this routine in other precisions, please see *slasd4*.

4.17.333 dlasd5

This subroutine computes the square root of the I-th eigenvalue of a positive symmetric rank-one modification of a 2-by-2 diagonal matrix

```
diag( D ) * diag( D ) +  RHO * Z * transpose(Z) .
```

The diagonal entries in the array D are assumed to satisfy

```
0 <= D(i) < D(j)  for  i < j .
```

We also assume RHO > 0 and that the Euclidean norm of the vector Z is one.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlasd5(I, D, Z, DELTA, RHO, DSIGMA, WORK)
```

C specification:

```
#include "armpl.h"

void dlasd5_(const armpl_int_t *i, const double *d, const double *z,
             double *delta, const double *rho, double *dsigma, double *work);
```

Parameters

I Input parameter.

I is INTEGER

The index of the eigenvalue to be computed. I = 1 or I = 2.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (2). The original eigenvalues. We assume 0 <= D(1) < D(2).

Z Input parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (2). The components of the updating vector.

DELTA Output parameter.

DELTA is DOUBLE PRECISION

DELTA is an array, dimension (2). Contains (D(j) - sigma_I) in its j-th component. The vector DELTA contains the information necessary to construct the eigenvectors.

RHO Input parameter.

RHO is DOUBLE PRECISION

The scalar in the symmetric updating formula.

DSIGMA Output parameter.

DSIGMA is DOUBLE PRECISION

The computed sigma_I, the I-th updated eigenvalue.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (2). WORK contains (D(j) + sigma_I) in its j-th component.

Related Information

For this routine in other precisions, please see *slasd5*.

4.17.334 dlasd6

dlasd6 computes the SVD of an updated upper bidiagonal matrix B obtained by merging two smaller ones by appending a row. This routine is used only for the problem which requires all singular values and optionally singular vector matrices in factored form. B is an N-by-M matrix with N = NL + NR + 1 and M = N + SQRE. A related subroutine, DLASD1, handles the case in which all singular values and singular vectors of the bidiagonal matrix are desired.

dlasd6 computes the SVD as follows:

```
              ( D1(in)     0     0        0 )
B = U(in) * (    Z1**T     a    Z2**T     b ) * VT(in)
              (    0        0    D2(in)    0 )

  = U(out) * ( D(out) 0) * VT(out)
```

where Z^T = ($Z1^T$ a $Z2^T$ b) = u^T VT^T , and u is a vector of dimension M with ALPHA and BETA in the NL+1 and NL+2 th entries and zeros elsewhere; and the entry b is empty if SQRE = 0.

The singular values of B can be computed using D1, D2, the first components of all the right singular vectors of the lower block, and the last components of all the right singular vectors of the upper block. These components are stored and updated in VF and VL, respectively, in dlasd6. Hence U and VT are not explicitly referenced.

The singular values are stored in D. The algorithm consists of two stages:

```
 The first stage consists of deflating the size of the problem
when there are multiple singular values or if there is a zero
in the Z vector. For each such occurrence the dimension of the
secular equation problem is reduced by one. This stage is
performed by the routine DLASD7.

The second stage consists of calculating the updated
singular values. This is done by finding the roots of the
secular equation via the routine DLASD4 (as called by DLASD8).
This routine also updates VF and VL and computes the distances
between the updated singular values and the old singular
values.
```

dlasd6 is called from DLASDA.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlasd6(ICOMPQ, NL, NR, SQRE, D, VF, VL, ALPHA, BETA, IDXQ, PERM,
                  GIVPTR, GIVCOL, LDGCOL, GIVNUM, LDGNUM, POLES, DIFL, DIFR,
                  Z, K, C, S, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dlasd6_(const armpl_int_t *icompq, const armpl_int_t *nl,
             const armpl_int_t *nr, const armpl_int_t *sqre, double *d,
             double *vf, double *vl, double *alpha, double *beta,
             armpl_int_t *idxq, armpl_int_t *perm, armpl_int_t *givptr,
             armpl_int_t *givcol, const armpl_int_t *ldgcol, double *givnum,
             const armpl_int_t *ldgnum, double *poles, double *difl,
             double *difr, double *z, armpl_int_t *k, double *c, double *s,
             double *work, armpl_int_t *iwork, armpl_int_t *info);
```

Parameters

ICOMPQ Input parameter.

ICOMPQ is INTEGER

Specifies whether singular vectors are to be computed in factored form: = 0: Compute singular values only. = 1: Compute singular vectors in factored form as well.

NL Input parameter.

NL is INTEGER

The row dimension of the upper block. NL >= 1.

NR Input parameter.

NR is INTEGER

The row dimension of the lower block. NR >= 1.

SQRE Input parameter.

SQRE is INTEGER

= 0: the lower block is an NR-by-NR square matrix. = 1: the lower block is an NR-by-(NR+1) rectangular matrix.

The bidiagonal matrix has row dimension N = NL + NR + 1, and column dimension M = N + SQRE.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (NL+NR+1).. On entry D(1:NL,1:NL) contains the singular values of the upper block, and D(NL+2:N) contains the singular values of the lower block. On exit D(1:N) contains the singular values of the modified matrix.

VF Input and output parameter.

VF is DOUBLE PRECISION

VF is an array, dimension (M). On entry, VF(1:NL+1) contains the first components of all right singular vectors of the upper block; and VF(NL+2:M) contains the first components of all right singular vectors of the lower block. On exit, VF contains the first components of all right singular vectors of the bidiagonal matrix.

VL Input and output parameter.

VL is DOUBLE PRECISION

VL is an array, dimension (M). On entry, VL(1:NL+1) contains the last components of all right singular vectors of the upper block; and VL(NL+2:M) contains the last components of all right singular vectors of the lower block. On exit, VL contains the last components of all right singular vectors of the bidiagonal matrix.

ALPHA Input and output parameter.

ALPHA is DOUBLE PRECISION

Contains the diagonal element associated with the added row.

BETA Input and output parameter.

BETA is DOUBLE PRECISION

Contains the off-diagonal element associated with the added row.

IDXQ Input and output parameter.

IDXQ is INTEGER array, dimension (N)

This contains the permutation which will reintegrate the subproblem just solved back into sorted order, i.e. D(IDXQ(I = 1, N)) will be in ascending order.

PERM Output parameter.

PERM is INTEGER array, dimension (N)

The permutations (from deflation and sorting) to be applied to each block. Not referenced if ICOMPQ = 0.

GIVPTR Output parameter.

GIVPTR is INTEGER

The number of Givens rotations which took place in this subproblem. Not referenced if ICOMPQ = 0.

GIVCOL Output parameter.

GIVCOL is INTEGER array, dimension (LDGCOL, 2)

Each pair of numbers indicates a pair of columns to take place in a Givens rotation. Not referenced if ICOMPQ = 0.

LDGCOL Input parameter.

LDGCOL is INTEGER

leading dimension of GIVCOL, must be at least N.

GIVNUM Output parameter.

GIVNUM is DOUBLE PRECISION

GIVNUM is an array, dimension (LDGNUM, 2). Each number indicates the C or S value to be used in the corresponding Givens rotation. Not referenced if ICOMPQ = 0.

LDGNUM Input parameter.

LDGNUM is INTEGER

The leading dimension of GIVNUM and POLES, must be at least N.

POLES Output parameter.

POLES is DOUBLE PRECISION

POLES is an array, dimension (LDGNUM, 2). On exit, POLES(1,*) is an array containing the new singular values obtained from solving the secular equation, and POLES(2,*) is an array containing the poles in the secular equation. Not referenced if ICOMPQ = 0.

DIFL Output parameter.

DIFL is DOUBLE PRECISION

DIFL is an array, dimension (N). On exit, DIFL(I) is the distance between I-th updated (undeflated) singular value and the I-th (undeflated) old singular value.

DIFR Output parameter.

DIFR is DOUBLE PRECISION

DIFR is an array,. dimension (LDDIFR, 2) if ICOMPQ = 1 and dimension (K) if ICOMPQ = 0. On exit, DIFR(I,1) = D(I) - DSIGMA(I+1), DIFR(K,1) is not defined and will not be referenced.

If ICOMPQ = 1, DIFR(1:K,2) is an array containing the normalizing factors for the right singular vector matrix.

See DLASD8 for details on DIFL and DIFR.

Z Output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (M). The first elements of this array contain the components of the deflation-adjusted updating row vector.

K Output parameter.

K is INTEGER

Contains the dimension of the non-deflated matrix, This is the order of the related secular equation. 1 <= K <=N.

C Output parameter.

C is DOUBLE PRECISION

C contains garbage if SQRE =0 and the C-value of a Givens rotation related to the right null space if SQRE = 1.

S Output parameter.

S is DOUBLE PRECISION

S contains garbage if SQRE =0 and the S-value of a Givens rotation related to the right null space if SQRE = 1.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (4 * M) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (3 * N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = 1, a singular value did not converge

Related Information

For this routine in other precisions, please see *slasd6*.

4.17.335 dlasd7

dlasd7 merges the two sets of singular values together into a single sorted set. Then it tries to deflate the size of the problem. There are two ways in which deflation can occur: when two or more singular values are close together or if there is a tiny entry in the Z vector. For each such occurrence the order of the related secular equation problem is reduced by one.

dlasd7 is called from DLASD6.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlasd7(ICOMPQ, NL, NR, SQRE, K, D, Z, ZW, VF, VFW, VL, VLW, ALPHA,
                  BETA, DSIGMA, IDX, IDXP, IDXQ, PERM, GIVPTR, GIVCOL, LDGCOL,
                  GIVNUM, LDGNUM, C, S, INFO)
```

C specification:

```
#include "armpl.h"

void dlasd7_(const armpl_int_t *icompq, const armpl_int_t *nl,
             const armpl_int_t *nr, const armpl_int_t *sqre, armpl_int_t *k,
             double *d, double *z, double *zw, double *vf, double *vfw,
             double *vl, double *vlw, const double *alpha, const double *beta,
             double *dsigma, armpl_int_t *idx, armpl_int_t *idxp,
             const armpl_int_t *idxq, armpl_int_t *perm, armpl_int_t *givptr,
             armpl_int_t *givcol, const armpl_int_t *ldgcol, double *givnum,
             const armpl_int_t *ldgnum, double *c, double *s,
             armpl_int_t *info);
```

Parameters

ICOMPQ Input parameter.

ICOMPQ is INTEGER

Specifies whether singular vectors are to be computed in compact form, as follows: = 0: Compute singular values only. = 1: Compute singular vectors of upper bidiagonal matrix in compact form.

NL Input parameter.

NL is INTEGER

The row dimension of the upper block. NL >= 1.

NR Input parameter.

NR is INTEGER

The row dimension of the lower block. NR >= 1.

SQRE Input parameter.

SQRE is INTEGER

= 0: the lower block is an NR-by-NR square matrix. = 1: the lower block is an NR-by-(NR+1) rectangular matrix.

The bidiagonal matrix has N = NL + NR + 1 rows and M = N + SQRE >= N columns.

K Output parameter.

K is INTEGER

Contains the dimension of the non-deflated matrix, this is the order of the related secular equation. 1 <= K <=N.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry D contains the singular values of the two submatrices to be combined. On exit D contains the trailing (N-K) updated singular values (those which were deflated) sorted into increasing order.

Z Output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (M). On exit Z contains the updating row vector in the secular equation.

ZW Output parameter.

ZW is DOUBLE PRECISION

ZW is an array, dimension (M). Workspace for Z.

VF Input and output parameter.

VF is DOUBLE PRECISION

VF is an array, dimension (M). On entry, VF(1:NL+1) contains the first components of all right singular vectors of the upper block; and VF(NL+2:M) contains the first components of all right singular vectors of the lower block. On exit, VF contains the first components of all right singular vectors of the bidiagonal matrix.

VFW Output parameter.

VFW is DOUBLE PRECISION

VFW is an array, dimension (M). Workspace for VF.

VL Input and output parameter.

VL is DOUBLE PRECISION

VL is an array, dimension (M). On entry, VL(1:NL+1) contains the last components of all right singular vectors of the upper block; and VL(NL+2:M) contains the last components of all right singular vectors of the lower block. On exit, VL contains the last components of all right singular vectors of the bidiagonal matrix.

VLW Output parameter.

VLW is DOUBLE PRECISION

VLW is an array, dimension (M). Workspace for VL.

ALPHA Input parameter.

ALPHA is DOUBLE PRECISION

Contains the diagonal element associated with the added row.

BETA Input parameter.

BETA is DOUBLE PRECISION

Contains the off-diagonal element associated with the added row.

DSIGMA Output parameter.

DSIGMA is DOUBLE PRECISION

DSIGMA is an array, dimension (N). Contains a copy of the diagonal elements (K-1 singular values and one zero) in the secular equation.

IDX Output parameter.

IDX is INTEGER array, dimension (N)

This will contain the permutation used to sort the contents of D into ascending order.

IDXP Output parameter.

IDXP is INTEGER array, dimension (N)

This will contain the permutation used to place deflated values of D at the end of the array. On output IDXP(2:K) points to the nondeflated D-values and IDXP(K+1:N) points to the deflated singular values.

IDXQ Input parameter.

IDXQ is INTEGER array, dimension (N)

This contains the permutation which separately sorts the two sub-problems in D into ascending order. Note that entries in the first half of this permutation must first be moved one position backward; and entries in the second half must first have NL+1 added to their values.

PERM Output parameter.

PERM is INTEGER array, dimension (N)

The permutations (from deflation and sorting) to be applied to each singular block. Not referenced if ICOMPQ = 0.

GIVPTR Output parameter.

GIVPTR is INTEGER

The number of Givens rotations which took place in this subproblem. Not referenced if ICOMPQ = 0.

GIVCOL Output parameter.

GIVCOL is INTEGER array, dimension (LDGCOL, 2)

Each pair of numbers indicates a pair of columns to take place in a Givens rotation. Not referenced if ICOMPQ = 0.

LDGCOL Input parameter.

LDGCOL is INTEGER

The leading dimension of GIVCOL, must be at least N.

GIVNUM Output parameter.

GIVNUM is DOUBLE PRECISION

GIVNUM is an array, dimension (LDGNUM, 2). Each number indicates the C or S value to be used in the corresponding Givens rotation. Not referenced if ICOMPQ = 0.

LDGNUM Input parameter.

LDGNUM is INTEGER

The leading dimension of GIVNUM, must be at least N.

C Output parameter.

C is DOUBLE PRECISION

C contains garbage if SQRE =0 and the C-value of a Givens rotation related to the right null space if SQRE = 1.

S Output parameter.

S is DOUBLE PRECISION

S contains garbage if SQRE =0 and the S-value of a Givens rotation related to the right null space if SQRE = 1.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *slasd7*.

4.17.336 dlasd8

dlasd8 finds the square roots of the roots of the secular equation, as defined by the values in DSIGMA and Z. It makes the appropriate calls to DLASD4, and stores, for each element in D, the distance to its two nearest poles (elements in DSIGMA). It also updates the arrays VF and VL, the first and last components of all the right singular vectors of the original bidiagonal matrix.

dlasd8 is called from DLASD6.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlasd8(ICOMPQ, K, D, Z, VF, VL, DIFL, DIFR, LDDIFR, DSIGMA, WORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dlasd8_(const armpl_int_t *icompq, const armpl_int_t *k, double *d,
             double *z, double *vf, double *vl, double *difl, double *difr,
             const armpl_int_t *lddifr, double *dsigma, double *work,
             armpl_int_t *info);
```

Parameters

ICOMPQ Input parameter.

ICOMPQ is INTEGER

Specifies whether singular vectors are to be computed in factored form in the calling routine: = 0: Compute singular values only. = 1: Compute singular vectors in factored form as well.

K Input parameter.

K is INTEGER

The number of terms in the rational function to be solved by DLASD4. K >= 1.

D Output parameter.

D is DOUBLE PRECISION

D is an array, dimension (K). On output, D contains the updated singular values.

Z Input and output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (K). On entry, the first K elements of this array contain the components of the deflation-adjusted updating row vector. On exit, Z is updated.

VF Input and output parameter.

VF is DOUBLE PRECISION

VF is an array, dimension (K). On entry, VF contains information passed through DBEDE8. On exit, VF contains the first K components of the first components of all right singular vectors of the bidiagonal matrix.

VL Input and output parameter.

VL is DOUBLE PRECISION

VL is an array, dimension (K). On entry, VL contains information passed through DBEDE8. On exit, VL contains the first K components of the last components of all right singular vectors of the bidiagonal matrix.

DIFL Output parameter.

DIFL is DOUBLE PRECISION

DIFL is an array, dimension (K). On exit, DIFL(I) = D(I) - DSIGMA(I).

DIFR Output parameter.

DIFR is DOUBLE PRECISION

DIFR is an array,. dimension (LDDIFR, 2) if ICOMPQ = 1 and dimension (K) if ICOMPQ = 0. On exit, DIFR(I,1) = D(I) - DSIGMA(I+1), DIFR(K,1) is not defined and will not be referenced.

If ICOMPQ = 1, DIFR(1:K,2) is an array containing the normalizing factors for the right singular vector matrix.

LDDIFR Input parameter.

LDDIFR is INTEGER

The leading dimension of DIFR, must be at least K.

DSIGMA Input and output parameter.

DSIGMA is DOUBLE PRECISION

DSIGMA is an array, dimension (K). On entry, the first K elements of this array contain the old roots of the deflated updating problem. These are the poles of the secular equation. On exit, the elements of DSIGMA may be very slightly altered in value.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (3*K) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = 1, a singular value did not converge

Related Information

For this routine in other precisions, please see *slasd8*.

4.17.337 dlasda

Using a divide and conquer approach, dlasda computes the singular value decomposition (SVD) of a real upper bidiagonal N-by-M matrix B with diagonal D and offdiagonal E, where M = N + SQRE. The algorithm computes the singular values in the SVD B = U * S * VT. The orthogonal matrices U and VT are optionally computed in compact form.

A related subroutine, DLASD0, computes the singular values and the singular vectors in explicit form.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlasda(ICOMPQ, SMLSIZ, N, SQRE, D, E, U, LDU, VT, K, DIFL, DIFR, Z,
                  POLES, GIVPTR, GIVCOL, LDGCOL, PERM, GIVNUM, C, S, WORK,
                  IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dlasda_(const armpl_int_t *icompq, const armpl_int_t *smlsiz,
             const armpl_int_t *n, const armpl_int_t *sqre, double *d,
             const double *e, double *u, const armpl_int_t *ldu, double *vt,
             armpl_int_t *k, double *difl, double *difr, double *z,
             double *poles, armpl_int_t *givptr, armpl_int_t *givcol,
             const armpl_int_t *ldgcol, armpl_int_t *perm, double *givnum,
             double *c, double *s, double *work, armpl_int_t *iwork,
             armpl_int_t *info);
```

Parameters

ICOMPQ Input parameter.

ICOMPQ is INTEGER

Specifies whether singular vectors are to be computed in compact form, as follows = 0: Compute singular values only. = 1: Compute singular vectors of upper bidiagonal matrix in compact form.

SMLSIZ Input parameter.

SMLSIZ is INTEGER

The maximum size of the subproblems at the bottom of the computation tree.

N Input parameter.

N is INTEGER

The row dimension of the upper bidiagonal matrix. This is also the dimension of the main diagonal array D.

SQRE Input parameter.

SQRE is INTEGER

Specifies the column dimension of the bidiagonal matrix. = 0: The bidiagonal matrix has column dimension M = N; = 1: The bidiagonal matrix has column dimension M = N + 1.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry D contains the main diagonal of the bidiagonal matrix. On exit D, if INFO = 0, contains its singular values.

E Input parameter.

E is DOUBLE PRECISION

E is an array, dimension (M-1). Contains the subdiagonal entries of the bidiagonal matrix. On exit, E has been destroyed.

U Output parameter.

U is DOUBLE PRECISION

U is an array,. dimension (LDU, SMLSIZ) if ICOMPQ = 1, and not referenced if ICOMPQ = 0. If ICOMPQ = 1, on exit, U contains the left singular vector matrices of all subproblems at the bottom level.

LDU Input parameter.

LDU is INTEGER, LDU = > N.

The leading dimension of arrays U, VT, DIFL, DIFR, POLES, GIVNUM, and Z.

VT Output parameter.

VT is DOUBLE PRECISION

VT is an array,. dimension (LDU, SMLSIZ+1) if ICOMPQ = 1, and not referenced if ICOMPQ = 0. If ICOMPQ = 1, on exit, VT^T contains the right singular vector matrices of all subproblems at the bottom level.

K Output parameter.

K is INTEGER array,

dimension (N) if ICOMPQ = 1 and dimension 1 if ICOMPQ = 0. If ICOMPQ = 1, on exit, K(I) is the dimension of the I-th secular equation on the computation tree.

DIFL Output parameter.

DIFL is DOUBLE PRECISION

DIFL is an array, dimension (LDU, NLVL),. where NLVL = floor(log_2 (N/SMLSIZ))).

DIFR Output parameter.

DIFR is DOUBLE PRECISION

DIFR is an array,. dimension (LDU, 2 * NLVL) if ICOMPQ = 1 and dimension (N) if ICOMPQ = 0. If ICOMPQ = 1, on exit, DIFL(1:N, I) and DIFR(1:N, 2 * I - 1) record distances between singular values on the I-th level and singular values on the (I -1)-th level, and DIFR(1:N, 2 * I) contains the normalizing factors for the right singular vector matrix. See DLASD8 for details.

Z Output parameter.

Z is DOUBLE PRECISION

Z is an array,. dimension (LDU, NLVL) if ICOMPQ = 1 and dimension (N) if ICOMPQ = 0. The first K elements of Z(1, I) contain the components of the deflation-adjusted updating row vector for subproblems on the I-th level.

POLES Output parameter.

POLES is DOUBLE PRECISION

POLES is an array,. dimension (LDU, 2 * NLVL) if ICOMPQ = 1, and not referenced if ICOMPQ = 0. If ICOMPQ = 1, on exit, POLES(1, 2*I - 1) and POLES(1, 2*I) contain the new and old singular values involved in the secular equations on the I-th level.

GIVPTR Output parameter.

GIVPTR is INTEGER array,

dimension (N) if ICOMPQ = 1, and not referenced if ICOMPQ = 0. If ICOMPQ = 1, on exit, GIVPTR(I) records the number of Givens rotations performed on the I-th problem on the computation tree.

GIVCOL Output parameter.

GIVCOL is INTEGER array,

dimension (LDGCOL, 2 * NLVL) if ICOMPQ = 1, and not referenced if ICOMPQ = 0. If ICOMPQ = 1, on exit, for each I, GIVCOL(1, 2 *I - 1) and GIVCOL(1, 2 *I) record the locations of Givens rotations performed on the I-th level on the computation tree.

LDGCOL Input parameter.

LDGCOL is INTEGER, LDGCOL = > N.

The leading dimension of arrays GIVCOL and PERM.

PERM Output parameter.

PERM is INTEGER array,

dimension (LDGCOL, NLVL) if ICOMPQ = 1, and not referenced if ICOMPQ = 0. If ICOMPQ = 1, on exit, PERM(1, I) records permutations done on the I-th level of the computation tree.

GIVNUM Output parameter.

GIVNUM is DOUBLE PRECISION

GIVNUM is an array,. dimension (LDU, 2 * NLVL) if ICOMPQ = 1, and not referenced if ICOMPQ = 0. If ICOMPQ = 1, on exit, for each I, GIVNUM(1, 2 *I - 1) and GIVNUM(1, 2 *I) record the C- and S- values of Givens rotations performed on the I-th level on the computation tree.

C Output parameter.

C is DOUBLE PRECISION

C is an array,. dimension (N) if ICOMPQ = 1, and dimension 1 if ICOMPQ = 0. If ICOMPQ = 1 and the I-th subproblem is not square, on exit, C(I) contains the C-value of a Givens rotation related to the right null space of the I-th subproblem.

S Output parameter.

S is DOUBLE PRECISION

S is an array, dimension (N) if. ICOMPQ = 1, and dimension 1 if ICOMPQ = 0. If ICOMPQ = 1 and the I-th subproblem is not square, on exit, S(I) contains the S-value of a Givens rotation related to the right null space of the I-th subproblem.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension. (6 * N + (SMLSIZ + 1)*(SMLSIZ + 1)).

IWORK Output parameter.

IWORK is INTEGER array, dimension (7*N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = 1, a singular value did not converge

Related Information

For this routine in other precisions, please see *slasda*.

4.17.338 dlasdq

dlasdq computes the singular value decomposition (SVD) of a real (upper or lower) bidiagonal matrix with diagonal D and offdiagonal E, accumulating the transformations if desired. Letting B denote the input bidiagonal matrix, the algorithm computes orthogonal matrices Q and P such that B = Q * S * P^T (P^T denotes the transpose of P). The singular values S are overwritten on D.

The input matrix U is changed to U * Q if desired. The input matrix VT is changed to P^T * VT if desired. The input matrix C is changed to Q^T * C if desired.

See "Computing Small Singular Values of Bidiagonal Matrices With Guaranteed High Relative Accuracy," by J. Demmel and W. Kahan, LAPACK Working Note #3, for a detailed description of the algorithm.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlasdq(UPLO, SQRE, N, NCVT, NRU, NCC, D, E, VT, LDVT, U, LDU, C,
                  LDC, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dlasdq_(const char *uplo, const armpl_int_t *sqre, const armpl_int_t *n,
             const armpl_int_t *ncvt, const armpl_int_t *nru,
             const armpl_int_t *ncc, double *d, double *e, double *vt,
             const armpl_int_t *ldvt, double *u, const armpl_int_t *ldu,
             double *c, const armpl_int_t *ldc, double *work,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the input bidiagonal matrix is upper or lower bidiagonal, and whether it is square are not. UPLO = 'U' or 'u' B is upper bidiagonal. UPLO = 'L' or 'l' B is lower bidiagonal.

SQRE Input parameter.

SQRE is INTEGER

= 0: then the input matrix is N-by-N. = 1: then the input matrix is N-by-(N+1) if UPLU = 'U' and (N+1)-by-N if UPLU = 'L'.

The bidiagonal matrix has N = NL + NR + 1 rows and M = N + SQRE >= N columns.

N Input parameter.

N is INTEGER

On entry, N specifies the number of rows and columns in the matrix. N must be at least 0.

NCVT Input parameter.

NCVT is INTEGER

On entry, NCVT specifies the number of columns of the matrix VT. NCVT must be at least 0.

NRU Input parameter.

NRU is INTEGER

On entry, NRU specifies the number of rows of the matrix U. NRU must be at least 0.

NCC Input parameter.

NCC is INTEGER

On entry, NCC specifies the number of columns of the matrix C. NCC must be at least 0.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry, D contains the diagonal entries of the bidiagonal matrix whose SVD is desired. On normal exit, D contains the singular values in ascending order.

E Input and output parameter.

E is DOUBLE PRECISION array.

dimension is (N-1) if SQRE = 0 and N if SQRE = 1. On entry, the entries of E contain the offdiagonal entries of the bidiagonal matrix whose SVD is desired. On normal exit, E will contain 0. If the algorithm does not converge, D and E will contain the diagonal and superdiagonal entries of a bidiagonal matrix orthogonally equivalent to the one given as input.

VT Input and output parameter.

VT is DOUBLE PRECISION

VT is an array, dimension (LDVT, NCVT). On entry, contains a matrix which on exit has been premultiplied by P^T , dimension N-by-NCVT if SQRE = 0 and (N+1)-by-NCVT if SQRE = 1 (not referenced if NCVT=0).

LDVT Input parameter.

LDVT is INTEGER

On entry, LDVT specifies the leading dimension of VT as declared in the calling (sub) program. LDVT must be at least 1. If NCVT is nonzero LDVT must also be at least N.

U Input and output parameter.

U is DOUBLE PRECISION

U is an array, dimension (LDU, N). On entry, contains a matrix which on exit has been postmultiplied by Q, dimension NRU-by-N if SQRE = 0 and NRU-by-(N+1) if SQRE = 1 (not referenced if NRU=0).

LDU Input parameter.

LDU is INTEGER

On entry, LDU specifies the leading dimension of U as declared in the calling (sub) program. LDU must be at least max(1, NRU) .

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, NCC). On entry, contains an N-by-NCC matrix which on exit has been premultiplied by Q^T dimension N-by-NCC if SQRE = 0 and (N+1)-by-NCC if SQRE = 1 (not referenced if NCC=0).

LDC Input parameter.

LDC is INTEGER

On entry, LDC specifies the leading dimension of C as declared in the calling (sub) program. LDC must be at least 1. If NCC is nonzero, LDC must also be at least N.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (4*N). Workspace. Only referenced if one of NCVT, NRU, or NCC is nonzero, and if N is at least 2.

INFO Output parameter.

INFO is INTEGER

On exit, a value of 0 indicates a successful exit. If INFO < 0, argument number -INFO is illegal. If INFO > 0, the algorithm did not converge, and INFO specifies how many superdiagonals did not converge.

Related Information

For this routine in other precisions, please see *slasdq*.

4.17.339 dlasdt

dlasdt creates a tree of subproblems for bidiagonal divide and conquer.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlasdt(N, LVL, ND, INODE, NDIML, NDIMR, MSUB)
```

C specification:

```
#include "armpl.h"

void dlasdt_(const armpl_int_t *n, armpl_int_t *lvl, armpl_int_t *nd,
             armpl_int_t *inode, armpl_int_t *ndiml, armpl_int_t *ndimr,
             const armpl_int_t *msub);
```

Parameters

N Input parameter.

N is INTEGER

On entry, the number of diagonal elements of the bidiagonal matrix.

LVL Output parameter.

LVL is INTEGER

On exit, the number of levels on the computation tree.

ND Output parameter.

ND is INTEGER

On exit, the number of nodes on the tree.

INODE Output parameter.

INODE is INTEGER array, dimension (N)

On exit, centers of subproblems.

NDIML Output parameter.

NDIML is INTEGER array, dimension (N)

On exit, row dimensions of left children.

NDIMR Output parameter.

NDIMR is INTEGER array, dimension (N)

On exit, row dimensions of right children.

MSUB Input parameter.

MSUB is INTEGER

On entry, the maximum row dimension each subproblem at the bottom of the tree can be of.

Related Information

For this routine in other precisions, please see *slasdt*.

4.17.340 dlaset

dlaset initializes an m-by-n matrix A to BETA on the diagonal and ALPHA on the offdiagonals.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlaset(UPLO, M, N, ALPHA, BETA, A, LDA)
```

C specification:

```
#include "armpl.h"

void dlaset_(const char *uplo, const armpl_int_t *m, const armpl_int_t *n,
             const double *alpha, const double *beta, double *a,
             const armpl_int_t *lda, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies the part of the matrix A to be set. = 'U': Upper triangular part is set; the strictly lower triangular part of A is not changed. = 'L': Lower triangular part is set; the strictly upper triangular part of A is not changed. Otherwise: All of the matrix A is set.

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

ALPHA Input parameter.

ALPHA is DOUBLE PRECISION

The constant to which the offdiagonal elements are to be set.

BETA Input parameter.

BETA is DOUBLE PRECISION

The constant to which the diagonal elements are to be set.

A Output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On exit, the leading m-by-n submatrix of A is set as follows:

if UPLO = 'U', A(i,j) = ALPHA, 1<=i<=j-1, 1<=j<=n, if UPLO = 'L', A(i,j) = ALPHA, j+1<=i<=m, 1<=j<=n, otherwise, A(i,j) = ALPHA, 1<=i<=m, 1<=j<=n, i.ne.j,

and, for all UPLO, A(i,i) = BETA, 1<=i<=min(m,n).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

Related Information

For this routine in other precisions, please see *claset*, *slaset* and *zlaset*. It also exists with a native C interface as *LAPACKE_dlaset*.

4.17.341 dlasq1

`dlasq1` computes the singular values of a real N-by-N bidiagonal matrix with diagonal D and off-diagonal E. The singular values are computed to high relative accuracy, in the absence of denormalization, underflow and overflow. The algorithm was first presented in

"Accurate singular values and differential qd algorithms" by K. V. Fernando and B. N. Parlett, Numer. Math., Vol-67, No. 2, pp. 191-230, 1994,

and the present implementation is described in "An implementation of the dqds Algorithm (Positive Case)", LAPACK Working Note.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlasq1(N, D, E, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dlasq1_(const armpl_int_t *n, double *d, double *e, double *work,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The number of rows and columns in the matrix. N >= 0.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry, D contains the diagonal elements of the bidiagonal matrix whose SVD is desired. On normal exit, D contains the singular values in decreasing order.

E Input and output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N). On entry, elements E(1:N-1) contain the off-diagonal elements of the bidiagonal matrix whose SVD is desired. On exit, E is overwritten.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (4*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: the algorithm failed = 1, a split was marked by a positive value in E = 2, current block of Z not diagonalized after 100*N iterations (in inner while loop) On exit D and E represent a matrix with the same singular values which the calling subroutine could use to finish the computation, or even feed back into DLASQ1 = 3, termination criterion of outer while loop not met (program created more than N unreduced blocks)

Related Information

For this routine in other precisions, please see *slasq1*.

4.17.342 dlasq2

dlasq2 computes all the eigenvalues of the symmetric positive definite tridiagonal matrix associated with the qd array Z to high relative accuracy are computed to high relative accuracy, in the absence of denormalization, underflow and overflow.

To see the relation of Z to the tridiagonal matrix, let L be a unit lower bidiagonal matrix with subdiagonals Z(2,4,6,,..) and let U be an upper bidiagonal matrix with 1's above and diagonal Z(1,3,5,,..). The tridiagonal is L*U or, if you prefer, the symmetric tridiagonal to which it is similar.

Note : dlasq2 defines a logical variable, IEEE, which is true on machines which follow ieee-754 floating-point standard in their handling of infinities and NaNs, and false otherwise. This variable is passed to DLASQ3.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlasq2(N, Z, INFO)
```

C specification:

```
#include "armpl.h"

void dlasq2_(const armpl_int_t *n, double *z, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The number of rows and columns in the matrix. N >= 0.

Z Input and output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (4* N). On entry Z holds the qd array. On exit, entries 1 to N hold the eigenvalues in decreasing order, Z(2*N+1) holds the trace, and Z(2*N+2) holds the sum of the eigenvalues. If N > 2, then Z(2*N+3) holds the iteration count, Z(2*N+4) holds NDIVS/NIN^2, and Z(2*N+5) holds the percentage of shifts that failed.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if the i-th argument is a scalar and had an illegal value, then INFO = -i, if the i-th argument is an array and the j-entry had an illegal value, then INFO = -(i*100+j) > 0: the algorithm failed = 1, a split was marked by a positive value in E = 2, current block of Z not diagonalized after 100*N iterations (in inner while loop). On exit Z holds a qd array with the same eigenvalues as the given Z. = 3, termination criterion of outer while loop not met (program created more than N unreduced blocks)

Related Information

For this routine in other precisions, please see *slasq2*.

4.17.343 dlasq3

dlasq3 checks for deflation, computes a shift (TAU) and calls dqds. In case of failure it changes shifts, and tries again until output is positive.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlasq3(I0, N0, Z, PP, DMIN, SIGMA, DESIG, QMAX, NFAIL, ITER, NDIV,
                  IEEE, TTYPE, DMIN1, DMIN2, DN, DN1, DN2, G, TAU)
```

C specification:

```
#include "armpl.h"

void dlasq3_(const armpl_int_t *i0, const armpl_int_t *n0, const double *z,
             armpl_int_t *pp, double *dmin, double *sigma, double *desig,
             const double *qmax, armpl_int_t *nfail, armpl_int_t *iter,
             armpl_int_t *ndiv, const armpl_int_t *ieee, armpl_int_t *ttype,
             double *dmin1, double *dmin2, double *dn, double *dn1,
             double *dn2, double *g, double *tau);
```

Parameters

I0 Input parameter.

I0 is INTEGER

First index.

N0 Input and output parameter.

N0 is INTEGER

Last index.

Z Input and output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (4* N0). Z holds the qd array.

PP Input and output parameter.

PP is INTEGER

PP=0 for ping, PP=1 for pong. PP=2 indicates that flipping was applied to the Z array and that the initial tests for deflation should not be performed.

DMIN Output parameter.

DMIN is DOUBLE PRECISION

Minimum value of d.

SIGMA Output parameter.

SIGMA is DOUBLE PRECISION

Sum of shifts used in current segment.

DESIG Input and output parameter.

DESIG is DOUBLE PRECISION

Lower order part of SIGMA

QMAX Input parameter.

QMAX is DOUBLE PRECISION

Maximum value of q.

NFAIL Input and output parameter.

NFAIL is INTEGER

Increment NFAIL by 1 each time the shift was too big.

ITER Input and output parameter.

ITER is INTEGER

Increment ITER by 1 for each iteration.

NDIV Input and output parameter.

NDIV is INTEGER

Increment NDIV by 1 for each division.

IEEE Input parameter.

IEEE is LOGICAL

Flag for IEEE or non IEEE arithmetic (passed to DLASQ5).

TTYPE Input and output parameter.

TTYPE is INTEGER

Shift type.

DMIN1 Input and output parameter.

DMIN1 is DOUBLE PRECISION

DMIN2 Input and output parameter.

DMIN2 is DOUBLE PRECISION

DN Input and output parameter.

DN is DOUBLE PRECISION

DN1 Input and output parameter.

DN1 is DOUBLE PRECISION

DN2 Input and output parameter.

DN2 is DOUBLE PRECISION

G Input and output parameter.

G is DOUBLE PRECISION

TAU Input and output parameter.

TAU is DOUBLE PRECISION

These are passed as arguments in order to save their values between calls to DLASQ3.

Related Information

For this routine in other precisions, please see *slasq3*.

4.17.344 dlasq4

dlasq4 computes an approximation TAU to the smallest eigenvalue using values of d from the previous transform.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlasq4(I0, N0, Z, PP, N0IN, DMIN, DMIN1, DMIN2, DN, DN1, DN2, TAU,
                  TTYPE, G)
```

C specification:

```
#include "armpl.h"

void dlasq4_(const armpl_int_t *i0, const armpl_int_t *n0, const double *z,
             const armpl_int_t *pp, armpl_int_t *n0in, const double *dmin,
             const double *dmin1, const double *dmin2, const double *dn,
             const double *dn1, const double *dn2, double *tau,
             armpl_int_t *ttype, double *g);
```

Parameters

I0 Input parameter.

I0 is INTEGER

First index.

N0 Input parameter.

N0 is INTEGER

Last index.

Z Input parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (4* N0). Z holds the qd array.

PP Input parameter.

PP is INTEGER

PP=0 for ping, PP=1 for pong.

N0IN Input parameter.

N0IN is INTEGER

The value of N0 at start of EIGTEST.

DMIN Input parameter.

DMIN is DOUBLE PRECISION

Minimum value of d.

DMIN1 Input parameter.

DMIN1 is DOUBLE PRECISION

Minimum value of d, excluding D(N0).

DMIN2 Input parameter.

DMIN2 is DOUBLE PRECISION

Minimum value of d, excluding D(N0) and D(N0-1).

DN Input parameter.

DN is DOUBLE PRECISION

d(N)

DN1 Input parameter.

DN1 is DOUBLE PRECISION

d(N-1)

DN2 Input parameter.

DN2 is DOUBLE PRECISION

d(N-2)

TAU Output parameter.

TAU is DOUBLE PRECISION

This is the shift.

TTYPE Output parameter.

TTYPE is INTEGER

Shift type.

G Input and output parameter.

G is DOUBLE PRECISION

G is passed as an argument in order to save its value between calls to DLASQ4.

Related Information

For this routine in other precisions, please see *slasq4*.

4.17.345 dlasq5

dlasq5 computes one dqds transform in ping-pong form, one version for IEEE machines another for non IEEE machines.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlasq5(I0, N0, Z, PP, TAU, SIGMA, DMIN, DMIN1, DMIN2, DN, DNM1,
                  DNM2, IEEE, EPS)
```

C specification:

```
#include "armpl.h"

void dlasq5_(const armpl_int_t *i0, const armpl_int_t *n0, const double *z,
             const armpl_int_t *pp, const double *tau, const double *sigma,
             double *dmin, double *dmin1, double *dmin2, double *dn,
             double *dnm1, double *dnm2, const armpl_int_t *ieee,
             const double *eps);
```

Parameters

I0 Input parameter.

I0 is INTEGER

First index.

N0 Input parameter.

N0 is INTEGER

Last index.

Z Input parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (4*N). Z holds the qd array. EMIN is stored in Z(4*N0) to avoid an extra argument.

PP Input parameter.

PP is INTEGER

PP=0 for ping, PP=1 for pong.

TAU Input parameter.

TAU is DOUBLE PRECISION

This is the shift.

SIGMA Input parameter.

SIGMA is DOUBLE PRECISION

This is the accumulated shift up to this step.

DMIN Output parameter.

DMIN is DOUBLE PRECISION

Minimum value of d.

DMIN1 Output parameter.

DMIN1 is DOUBLE PRECISION

Minimum value of d, excluding D(N0).

DMIN2 Output parameter.

DMIN2 is DOUBLE PRECISION

Minimum value of d, excluding D(N0) and D(N0-1).

DN Output parameter.

DN is DOUBLE PRECISION

d(N0), the last value of d.

DNM1 Output parameter.

DNM1 is DOUBLE PRECISION

d(N0-1).

DNM2 Output parameter.

DNM2 is DOUBLE PRECISION

d(N0-2).

IEEE Input parameter.

IEEE is LOGICAL

Flag for IEEE or non IEEE arithmetic.

EPS Input parameter.

EPS is DOUBLE PRECISION

This is the value of epsilon used.

Related Information

For this routine in other precisions, please see *slasq5*.

4.17.346 dlasq6

dlasq6 computes one dqd (shift equal to zero) transform in ping-pong form, with protection against underflow and overflow.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlasq6(I0, N0, Z, PP, DMIN, DMIN1, DMIN2, DN, DNM1, DNM2)
```

C specification:

```
#include "armpl.h"

void dlasq6_(const armpl_int_t *i0, const armpl_int_t *n0, const double *z,
             const armpl_int_t *pp, double *dmin, double *dmin1,
             double *dmin2, double *dn, double *dnm1, double *dnm2);
```

Parameters

I0 Input parameter.

I0 is INTEGER

First index.

N0 Input parameter.

N0 is INTEGER

Last index.

Z Input parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (4*N). Z holds the qd array. EMIN is stored in Z(4*N0) to avoid an extra argument.

PP Input parameter.

PP is INTEGER

PP=0 for ping, PP=1 for pong.

DMIN Output parameter.

DMIN is DOUBLE PRECISION

Minimum value of d.

DMIN1 Output parameter.

DMIN1 is DOUBLE PRECISION

Minimum value of d, excluding D(N0).

DMIN2 Output parameter.

DMIN2 is DOUBLE PRECISION

Minimum value of d, excluding D(N0) and D(N0-1).

DN Output parameter.

DN is DOUBLE PRECISION

d(N0), the last value of d.

DNM1 Output parameter.

DNM1 is DOUBLE PRECISION

d(N0-1).

DNM2 Output parameter.

DNM2 is DOUBLE PRECISION

d(N0-2).

Related Information

For this routine in other precisions, please see *slasq6*.

4.17.347 dlasr

`dlasr` applies a sequence of plane rotations to a real matrix A, from either the left or the right.

When SIDE = 'L', the transformation takes the form

```
A := P*A
```

and when SIDE = 'R', the transformation takes the form

```
A := A*P**T
```

where P is an orthogonal matrix consisting of a sequence of z plane rotations, with z = M when SIDE = 'L' and z = N when SIDE = 'R', and P^T is the transpose of P.

When DIRECT = 'F' (Forward sequence), then

```
P = P(z-1) * ... * P(2) * P(1)
```

and when DIRECT = 'B' (Backward sequence), then

```
P = P(1) * P(2) * ... * P(z-1)
```

where P(k) is a plane rotation matrix defined by the 2-by-2 rotation

```
R(k) = (  c(k)  s(k) )
     = ( -s(k)  c(k) ).
```

When PIVOT = 'V' (Variable pivot), the rotation is performed for the plane (k,k+1), i.e., P(k) has the form

```
P(k) = (  1                                          )
       (       ...                                   )
       (              1                              )
       (                    c(k)  s(k)               )
       (                   -s(k)  c(k)               )
       (                             1               )
       (                                  ...        )
       (                                        1    )
```

where R(k) appears as a rank-2 modification to the identity matrix in rows and columns k and k+1.

When PIVOT = 'T' (Top pivot), the rotation is performed for the plane (1,k+1), so P(k) has the form

```
P(k) = (  c(k)                    s(k)               )
       (         1                                   )
       (              ...                            )
       (                     1                       )
       ( -s(k)                    c(k)               )
       (                                 1           )
       (                                      ...    )
       (                                            1 )
```

where R(k) appears in rows and columns 1 and k+1.

Similarly, when PIVOT = 'B' (Bottom pivot), the rotation is performed for the plane (k,z), giving P(k) the form

```
P(k) = ( 1                                           )
       (       ...                                   )
       (              1                              )
       (                    c(k)               s(k)  )
       (                           1                )
       (                                ...         )
       (                                     1      )
       (                   -s(k)               c(k)  )
```

where R(k) appears in rows and columns k and z. The rotations are performed without ever forming P(k) explicitly.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlasr(SIDE, PIVOT, DIRECT, M, N, C, S, A, LDA)
```

C specification:

```
#include "armpl.h"

void dlasr_(const char *side, const char *pivot, const char *direct,
            const armpl_int_t *m, const armpl_int_t *n, const double *c,
            const double *s, double *a, const armpl_int_t *lda, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

Specifies whether the plane rotation matrix P is applied to A on the left or the right. = 'L': Left, compute A := P*A = 'R': Right, compute A:= A*P^T

PIVOT Input parameter.

PIVOT is CHARACTER*1

Specifies the plane for which P(k) is a plane rotation matrix. = 'V': Variable pivot, the plane (k,k+1) = 'T': Top pivot, the plane (1,k+1) = 'B': Bottom pivot, the plane (k,z)

DIRECT Input parameter.

DIRECT is CHARACTER*1

Specifies whether P is a forward or backward sequence of plane rotations. = 'F': Forward, P = P(z-1)*...*P(2)*P(1) = 'B': Backward, P = P(1)*P(2)*...*P(z-1)

M Input parameter.

M is INTEGER

The number of rows of the matrix A. If m <= 1, an immediate return is effected.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. If n <= 1, an immediate return is effected.

C Input parameter.

C is DOUBLE PRECISION

C is an array, dimension. (M-1) if SIDE = 'L' (N-1) if SIDE = 'R' The cosines c(k) of the plane rotations.

S Input parameter.

S is DOUBLE PRECISION

S is an array, dimension. (M-1) if SIDE = 'L' (N-1) if SIDE = 'R' The sines s(k) of the plane rotations. The 2-by-2 plane rotation part of the matrix P(k), R(k), has the form R(k) = (c(k) s(k)) (-s(k) c(k)).

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). The M-by-N matrix A. On exit, A is overwritten by P* A if SIDE = 'R' or by A*P^T if SIDE = 'L'.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

Related Information

For this routine in other precisions, please see *clasr*, *slasr* and *zlasr*.

4.17.348 dlasrt

Sort the numbers in D in increasing order (if ID = 'I') or in decreasing order (if ID = 'D').

Use Quick Sort, reverting to Insertion sort on arrays of size <= 20. Dimension of STACK limits N to about 2**32.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlasrt(ID, N, D, INFO)
```

C specification:

```
#include "armpl.h"

void dlasrt_(const char *id, const armpl_int_t *n, double *d,
             armpl_int_t *info, ... );
```

Parameters

ID Input parameter.

ID is CHARACTER*1

= 'I': sort D in increasing order; = 'D': sort D in decreasing order.

N Input parameter.

N is INTEGER

The length of the array D.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry, the array to be sorted. On exit, D has been sorted into increasing order (D(1) <= ... <= D(N)) or into decreasing order (D(1) >= ... >= D(N)), depending on ID.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *slasrt*. It also exists with a native C interface as *LAPACKE_dlasrt*.

4.17.349 dlassq

dlassq returns the values scl and smsq such that

```
( scl**2 )*smsq = x( 1 )**2 +...+ x( n )**2 + ( scale**2 )*sumsq,
```

where x(i) = X(1 + (i - 1)*INCX). The value of sumsq is assumed to be non-negative and scl returns the value

```
scl = max( scale, abs( x( i ) ) ).
```

scale and sumsq must be supplied in SCALE and SUMSQ and scl and smsq are overwritten on SCALE and SUMSQ respectively.

The routine makes only one pass through the vector x.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlassq(N, X, INCX, SCALE, SUMSQ)
```

C specification:

```
#include "armpl.h"

void dlassq_(const armpl_int_t *n, const double *x, const armpl_int_t *incx,
             double *scale, double *sumsq);
```

Parameters

N Input parameter.

N is INTEGER

The number of elements to be used from the vector X.

X Input parameter.

X is DOUBLE PRECISION

X is an array, dimension (N). The vector for which a scaled sum of squares is computed. x(i) = X(1 + (i - 1)* INCX), 1 <= i <= n.

INCX Input parameter.

INCX is INTEGER

The increment between successive values of the vector X. INCX > 0.

SCALE Input and output parameter.

SCALE is DOUBLE PRECISION

On entry, the value scale in the equation above. On exit, SCALE is overwritten with scl , the scaling factor for the sum of squares.

SUMSQ Input and output parameter.

SUMSQ is DOUBLE PRECISION

On entry, the value sumsq in the equation above. On exit, SUMSQ is overwritten with smsq , the basic sum of squares from which scl has been factored out.

Related Information

For this routine in other precisions, please see *classq*, *slassq* and *zlassq*. It also exists with a native C interface as *LAPACKE_dlassq*.

4.17.350 dlasv2

dlasv2 computes the singular value decomposition of a 2-by-2 triangular matrix

```
[  F   G  ]
[  0   H  ].
```

On return, abs(SSMAX) is the larger singular value, abs(SSMIN) is the smaller singular value, and (CSL,SNL) and (CSR,SNR) are the left and right singular vectors for abs(SSMAX), giving the decomposition

```
[ CSL  SNL ] [  F   G  ] [ CSR -SNR ]  =  [ SSMAX   0   ]
[-SNL  CSL ] [  0   H  ] [ SNR  CSR ]     [  0    SSMIN ].
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dlasv2(F, G, H, SSMIN, SSMAX, SNR, CSR, SNL, CSL)
```

C specification:

```
#include "armpl.h"

void dlasv2_(const double *f, const double *g, const double *h, double *ssmin,
             double *ssmax, double *snr, double *csr, double *snl,
             double *csl);
```

Parameters

F Input parameter.

F is DOUBLE PRECISION

The (1,1) element of the 2-by-2 matrix.

G Input parameter.

G is DOUBLE PRECISION

The (1,2) element of the 2-by-2 matrix.

H Input parameter.

H is DOUBLE PRECISION

The (2,2) element of the 2-by-2 matrix.

SSMIN Output parameter.

SSMIN is DOUBLE PRECISION

abs(SSMIN) is the smaller singular value.

SSMAX Output parameter.

SSMAX is DOUBLE PRECISION

abs(SSMAX) is the larger singular value.

SNL Output parameter.

SNL is DOUBLE PRECISION

CSL Output parameter.

CSL is DOUBLE PRECISION

The vector (CSL, SNL) is a unit left singular vector for the singular value abs(SSMAX).

SNR Output parameter.

SNR is DOUBLE PRECISION

CSR Output parameter.

CSR is DOUBLE PRECISION

The vector (CSR, SNR) is a unit right singular vector for the singular value abs(SSMAX).

Related Information

For this routine in other precisions, please see *slasv2*.

4.17.351 dlaswlq

dlaswlq computes a blocked Short-Wide LQ factorization of a M-by-N matrix A, where N >= M: A = L * Q

Syntax

Fortran specification:

```
use armpl_library

subroutine dlaswlq(M, N, MB, NB, A, LDA, T, LDT, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dlaswlq_(const armpl_int_t *m, const armpl_int_t *n,
              const armpl_int_t *mb, const armpl_int_t *nb, double *a,
              const armpl_int_t *lda, double *t, const armpl_int_t *ldt,
              double *work, const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= M >= 0.

MB Input parameter.

MB is INTEGER

The row block size to be used in the blocked QR. M >= MB >= 1

NB Input parameter.

NB is INTEGER

The column block size to be used in the blocked QR. NB > M.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and below the diagonal of the array contain the N-by-N lower triangular matrix L; the elements above the diagonal represent Q by the rows of blocked V (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

T Output parameter.

T is DOUBLE PRECISION

T is an array,. dimension (LDT, N * Number_of_row_blocks) where Number_of_row_blocks = CEIL((N-M)/(NB-M)) The blocked upper triangular block reflectors stored in compact form as a sequence of upper triangular blocks. See Further Details below.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= MB.

WORK Output parameter.

(workspace) DOUBLE PRECISION

(workspace) is an array, dimension (MAX(1, LWORK)) .

LWORK Input parameter.

The dimension of the array WORK. LWORK >= MB*M.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *claswlq*, *slaswlq* and *zlaswlq*.

4.17.352 dlaswp

`dlaswp` performs a series of row interchanges on the matrix A. One row interchange is initiated for each of rows K1 through K2 of A.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlaswp(N, A, LDA, K1, K2, IPIV, INCX)
```

C specification:

```
#include "armpl.h"

void dlaswp_(const armpl_int_t *n, double *a, const armpl_int_t *lda,
             const armpl_int_t *k1, const armpl_int_t *k2,
             const armpl_int_t *ipiv, const armpl_int_t *incx);
```

Parameters

N Input parameter.

N is INTEGER

The number of columns of the matrix A.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the matrix of column dimension N to which the row interchanges will be applied. On exit, the permuted matrix.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A.

K1 Input parameter.

K1 is INTEGER

The first element of IPIV for which a row interchange will be done.

K2 Input parameter.

K2 is INTEGER

(K2-K1+1) is the number of elements of IPIV for which a row interchange will be done.

IPIV Input parameter.

IPIV is INTEGER array, dimension (K1+(K2-K1)*abs(INCX))

The vector of pivot indices. Only the elements in positions K1 through K1+(K2-K1)*abs(INCX) of IPIV are accessed. IPIV(K1+(K-K1)*abs(INCX)) = L implies rows K and L are to be interchanged.

INCX Input parameter.

INCX is INTEGER

The increment between successive values of IPIV. If INCX is negative, the pivots are applied in reverse order.

Related Information

For this routine in other precisions, please see *claswp*, *slaswp* and *zlaswp*. It also exists with a native C interface as *LAPACKE_dlaswp*.

4.17.353 dlasy2

`dlasy2` solves for the N1 by N2 matrix X, 1 <= N1,N2 <= 2, in

```
op(TL)*X + ISGN*X*op(TR) = SCALE*B,
```

where TL is N1 by N1, TR is N2 by N2, B is N1 by N2, and ISGN = 1 or -1. op(T) = T or T^T , where T^T denotes the transpose of T.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlasy2(LTRANL, LTRANR, ISGN, N1, N2, TL, LDTL, TR, LDTR, B, LDB,
                  SCALE, X, LDX, XNORM, INFO)
```

C specification:

```
#include "armpl.h"

void dlasy2_(const armpl_int_t *ltranl, const armpl_int_t *ltranr,
             const armpl_int_t *isgn, const armpl_int_t *n1,
             const armpl_int_t *n2, const double *tl, const armpl_int_t *ldtl,
             const double *tr, const armpl_int_t *ldtr, const double *b,
             const armpl_int_t *ldb, double *scale, double *x,
             const armpl_int_t *ldx, double *xnorm, armpl_int_t *info);
```

Parameters

LTRANL Input parameter.

LTRANL is LOGICAL

On entry, LTRANL specifies the op(TL): = .FALSE., op(TL) = TL, = .TRUE., op(TL) = TL^T .

LTRANR Input parameter.

LTRANR is LOGICAL

On entry, LTRANR specifies the op(TR): = .FALSE., op(TR) = TR, = .TRUE., op(TR) = TR^T .

ISGN Input parameter.

ISGN is INTEGER

On entry, ISGN specifies the sign of the equation as described before. ISGN may only be 1 or -1.

N1 Input parameter.

N1 is INTEGER

On entry, N1 specifies the order of matrix TL. N1 may only be 0, 1 or 2.

N2 Input parameter.

N2 is INTEGER

On entry, N2 specifies the order of matrix TR. N2 may only be 0, 1 or 2.

TL Input parameter.

TL is DOUBLE PRECISION

TL is an array, dimension (LDTL,2). On entry, TL contains an N1 by N1 matrix.

LDTL Input parameter.

LDTL is INTEGER

The leading dimension of the matrix TL. LDTL >= max(1, N1).

TR Input parameter.

TR is DOUBLE PRECISION

TR is an array, dimension (LDTR,2). On entry, TR contains an N2 by N2 matrix.

LDTR Input parameter.

LDTR is INTEGER

The leading dimension of the matrix TR. LDTR >= max(1, N2).

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB,2). On entry, the N1 by N2 matrix B contains the right-hand side of the equation.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the matrix B. LDB >= max(1, N1).

SCALE Output parameter.

SCALE is DOUBLE PRECISION

On exit, SCALE contains the scale factor. SCALE is chosen less than or equal to 1 to prevent the solution overflowing.

X Output parameter.

X is DOUBLE PRECISION

X is an array, dimension (LDX,2). On exit, X contains the N1 by N2 solution.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the matrix X. LDX >= max(1, N1).

XNORM Output parameter.

XNORM is DOUBLE PRECISION

On exit, XNORM is the infinity-norm of the solution.

INFO Output parameter.

INFO is INTEGER

On exit, INFO is set to 0: successful exit. 1: TL and TR have too close eigenvalues, so TL or TR is perturbed to get a nonsingular equation. NOTE: In the interests of speed, this routine does not check the inputs for errors.

Related Information

For this routine in other precisions, please see *slasy2*.

4.17.354 dlasyf

`dlasyf` computes a partial factorization of a real symmetric matrix A using the Bunch-Kaufman diagonal pivoting method. The partial factorization has the form:

A = (I U12) (A11 0) (I 0) if UPLO = 'U', or:

```
( 0  U22 ) (  0   D  ) ( U12**T U22**T )
```

A = (L11 0) (D 0) ($L11^T$ $L21^T$) if UPLO = 'L'

```
( L21  I ) (  0  A22 ) (  0       I    )
```

where the order of D is at most NB. The actual order is returned in the argument KB, and is either NB or NB-1, or N if N <= NB.

`dlasyf` is an auxiliary routine called by DSYTRF. It uses blocked code (calling Level 3 BLAS) to update the submatrix A11 (if UPLO = 'U') or A22 (if UPLO = 'L').

Syntax

Fortran specification:

```
use armpl_library

subroutine dlasyf(UPLO, N, NB, KB, A, LDA, IPIV, W, LDW, INFO)
```

C specification:

```
#include "armpl.h"

void dlasyf_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nb,
             armpl_int_t *kb, double *a, const armpl_int_t *lda,
             armpl_int_t *ipiv, double *w, const armpl_int_t *ldw,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NB Input parameter.

NB is INTEGER

The maximum number of columns of the matrix A that should be factored. NB should be at least 2 to allow for 2-by-2 pivot blocks.

KB Output parameter.

KB is INTEGER

The number of columns of A that were actually factored. KB is either NB-1 or NB, or N if N <= NB.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, A contains details of the partial factorization.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D.

If UPLO = 'U': Only the last KB elements of IPIV are set.

If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block.

If UPLO = 'L': Only the first KB elements of IPIV are set.

If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (LDW, NB) .

LDW Input parameter.

LDW is INTEGER

The leading dimension of the array W. LDW >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit > 0: if INFO = k, D(k,k) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular.

Related Information

For this routine in other precisions, please see *clasyf*, *slasyf* and *zlasyf*.

4.17.355 dlasyf_aa

DLATRF_AA factorizes a panel of a real symmetric matrix A using the Aasen's algorithm. The panel consists of a set of NB rows of A when UPLO is U, or a set of NB columns when UPLO is L.

In order to factorize the panel, the Aasen's algorithm requires the last row, or column, of the previous panel. The first row, or column, of A is set to be the first row, or column, of an identity matrix, which is used to factorize the first panel.

The resulting J-th row of U, or J-th column of L, is stored in the (J-1)-th row, or column, of A (without the unit diagonals), while the diagonal and subdiagonal of A are overwritten by those of T.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlasyf_aa(UPLO, J1, M, NB, A, LDA, IPIV, H, LDH, WORK)
```

C specification:

```
#include "armpl.h"

void dlasyf_aa_(const char *uplo, const armpl_int_t *j1, const armpl_int_t *m,
                const armpl_int_t *nb, double *a, const armpl_int_t *lda,
                armpl_int_t *ipiv, double *h, const armpl_int_t *ldh,
                double *work, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

J1 Input parameter.

J1 is INTEGER

The location of the first row, or column, of the panel within the submatrix of A, passed to this routine, e.g., when called by DSYTRF_AA, for the first panel, J1 is 1, while for the remaining panels, J1 is 2.

M Input parameter.

M is INTEGER

The dimension of the submatrix. M >= 0.

NB Input parameter.

NB is INTEGER

The dimension of the panel to be facotorized.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, M) for. the first panel, while dimension (LDA,M+1) for the remaining panels.

On entry, A contains the last row, or column, of the previous panel, and the trailing submatrix of A to be factorized, except for the first panel, only the panel is passed.

On exit, the leading panel is factorized.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

IPIV Output parameter.

IPIV is INTEGER array, dimension (M)

Details of the row and column interchanges, the row and column k were interchanged with the row and column IPIV(k).

H Input and output parameter.

H is DOUBLE PRECISION workspace, dimension (LDH, NB).

LDH Input parameter.

LDH is INTEGER

The leading dimension of the workspace H. LDH >= max(1, M).

WORK Output parameter.

WORK is DOUBLE PRECISION workspace, dimension (M).

Related Information

For this routine in other precisions, please see *clasyf_aa*, *slasyf_aa* and *zlasyf_aa*.

4.17.356 dlasyf_rook

DLASYF_ROOK computes a partial factorization of a real symmetric matrix A using the bounded Bunch-Kaufman ("rook") diagonal pivoting method. The partial factorization has the form:

A = (I U12) (A11 0) (I 0) if UPLO = 'U', or:

```
( 0  U22 ) (  0   D  ) ( U12**T U22**T )
```

A = (L11 0) (D 0) ($L11^T$ $L21^T$) if UPLO = 'L'

```
( L21  I ) (  0  A22 ) (  0       I    )
```

where the order of D is at most NB. The actual order is returned in the argument KB, and is either NB or NB-1, or N if N <= NB.

DLASYF_ROOK is an auxiliary routine called by DSYTRF_ROOK. It uses blocked code (calling Level 3 BLAS) to update the submatrix A11 (if UPLO = 'U') or A22 (if UPLO = 'L').

Syntax

Fortran specification:

```
use armpl_library

subroutine dlasyf_rook(UPLO, N, NB, KB, A, LDA, IPIV, W, LDW, INFO)
```

C specification:

```
#include "armpl.h"

void dlasyf_rook_(const char *uplo, const armpl_int_t *n,
                  const armpl_int_t *nb, armpl_int_t *kb, double *a,
                  const armpl_int_t *lda, armpl_int_t *ipiv, double *w,
                  const armpl_int_t *ldw, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NB Input parameter.

NB is INTEGER

The maximum number of columns of the matrix A that should be factored. NB should be at least 2 to allow for 2-by-2 pivot blocks.

KB Output parameter.

KB is INTEGER

The number of columns of A that were actually factored. KB is either NB-1 or NB, or N if N <= NB.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, A contains details of the partial factorization.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D.

If UPLO = 'U': Only the last KB elements of IPIV are set.

If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k-1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k-1 and -IPIV(k-1) were inerchaged, D(k-1:k,k-1:k) is a 2-by-2 diagonal block.

If UPLO = 'L': Only the first KB elements of IPIV are set.

If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k+1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k+1 and -IPIV(k+1) were inerchaged, D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (LDW, NB) .

LDW Input parameter.

LDW is INTEGER

The leading dimension of the array W. LDW >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit > 0: if INFO = k, D(k,k) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular.

Related Information

For this routine in other precisions, please see *clasyf_rook*, *slasyf_rook* and *zlasyf_rook*.

4.17.357 dlat2s

dlat2s converts a DOUBLE PRECISION triangular matrix, SA, to a SINGLE PRECISION triangular matrix, A.

RMAX is the overflow for the SINGLE PRECISION arithmetic DLAS2S checks that all the entries of A are between -RMAX and RMAX. If not the conversion is aborted and a flag is raised.

This is an auxiliary routine so there is no argument checking.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlat2s(UPLO, N, A, LDA, SA, LDSA, INFO)
```

C specification:

```
#include "armpl.h"

void dlat2s_(const char *uplo, const armpl_int_t *n, const double *a,
             const armpl_int_t *lda, float *sa, const armpl_int_t *ldsa,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

N Input parameter.

N is INTEGER

The number of rows and columns of the matrix A. N >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the N-by-N triangular coefficient matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

SA Output parameter.

SA is REAL

SA is an array, dimension (LDSA, N). Only the UPLO part of SA is referenced. On exit, if INFO=0, the N-by-N coefficient matrix SA; if INFO>0, the content of the UPLO part of SA is unspecified.

LDSA Input parameter.

LDSA is INTEGER

The leading dimension of the array SA. LDSA >= max(1,M).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. = 1: an entry of the matrix A is greater than the SINGLE PRECISION overflow threshold, in this case, the content of the UPLO part of SA in exit is unspecified.

Related Information

4.17.358 dlatbs

dlatbs solves one of the triangular systems

```
A *x = s*b  or  A**T*x = s*b
```

with scaling to prevent overflow, where A is an upper or lower triangular band matrix. Here A^T denotes the transpose of A, x and b are n-element vectors, and s is a scaling factor, usually less than or equal to 1, chosen so that the components of x will be less than the overflow threshold. If the unscaled problem will not cause overflow, the Level 2 BLAS routine DTBSV is called. If the matrix A is singular (A(j,j) = 0 for some j), then s is set to 0 and a non-trivial solution to A*x = 0 is returned.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlatbs(UPLO, TRANS, DIAG, NORMIN, N, KD, AB, LDAB, X, SCALE, CNORM,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dlatbs_(const char *uplo, const char *trans, const char *diag,
             const char *normin, const armpl_int_t *n, const armpl_int_t *kd,
             const double *ab, const armpl_int_t *ldab, double *x,
             double *scale, double *cnorm, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the matrix A is upper or lower triangular. = 'U': Upper triangular = 'L': Lower triangular

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the operation applied to A. = 'N': Solve A * x = s*b (No transpose) = 'T': Solve A^T * x = s*b (Transpose) = 'C': Solve A^T * x = s*b (Conjugate transpose = Transpose)

DIAG Input parameter.

DIAG is CHARACTER*1

Specifies whether or not the matrix A is unit triangular. = 'N': Non-unit triangular = 'U': Unit triangular

NORMIN Input parameter.

NORMIN is CHARACTER*1

Specifies whether CNORM has been set or not. = 'Y': CNORM contains the column norms on entry = 'N': CNORM is not set on entry. On exit, the norms will be computed and stored in CNORM.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of subdiagonals or superdiagonals in the triangular matrix A. KD >= 0.

AB Input parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). The upper or lower triangular band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

X Input and output parameter.

X is DOUBLE PRECISION

X is an array, dimension (N). On entry, the right hand side b of the triangular system. On exit, X is overwritten by the solution vector x.

SCALE Output parameter.

SCALE is DOUBLE PRECISION

The scaling factor s for the triangular system A * x = s*b or A^T * x = s*b. If SCALE = 0, the matrix A is singular or badly scaled, and the vector x is an exact or approximate solution to A*x = 0.

CNORM Input and output parameter.

CNORM is DOUBLE PRECISION

CNORM is an array, dimension (N). If NORMIN = 'Y', CNORM is an input argument and CNORM(j) contains the norm of the off-diagonal part of the j-th column of A. If TRANS = 'N', CNORM(j) must be greater than

or equal to the infinity-norm, and if TRANS = 'T' or 'C', CNORM(j) must be greater than or equal to the 1-norm.

If NORMIN = 'N', CNORM is an output argument and CNORM(j) returns the 1-norm of the offdiagonal part of the j-th column of A.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value

Related Information

For this routine in other precisions, please see *clatbs*, *slatbs* and *zlatbs*.

4.17.359 dlatdf

dlatdf uses the LU factorization of the n-by-n matrix Z computed by DGETC2 and computes a contribution to the reciprocal Dif-estimate by solving Z * x = b for x, and choosing the r.h.s. b such that the norm of x is as large as possible. On entry RHS = b holds the contribution from earlier solved sub-systems, and on return RHS = x.

The factorization of Z returned by DGETC2 has the form Z = P*L*U*Q, where P and Q are permutation matrices. L is lower triangular with unit diagonal elements and U is upper triangular.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlatdf(IJOB, N, Z, LDZ, RHS, RDSUM, RDSCAL, IPIV, JPIV)
```

C specification:

```
#include "armpl.h"

void dlatdf_(const armpl_int_t *ijob, const armpl_int_t *n, const double *z,
             const armpl_int_t *ldz, double *rhs, double *rdsum,
             double *rdscal, const armpl_int_t *ipiv,
             const armpl_int_t *jpiv);
```

Parameters

IJOB Input parameter.

IJOB is INTEGER

IJOB = 2: First compute an approximative null-vector e of Z using DGECON, e is normalized and solve for Zx = +-e - f with the sign giving the greater value of 2-norm(x). About 5 times as expensive as Default. IJOB .ne. 2: Local look ahead strategy where all entries of the r.h.s. b is chosen as either +1 or -1 (Default).

N Input parameter.

N is INTEGER

The number of columns of the matrix Z.

Z Input parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, N). On entry, the LU part of the factorization of the n-by-n matrix Z computed by DGETC2: Z = P * L * U * Q

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDA >= max(1, N).

RHS Input and output parameter.

RHS is DOUBLE PRECISION

RHS is an array, dimension (N). On entry, RHS contains contributions from other subsystems. On exit, RHS contains the solution of the subsystem with entries acoording to the value of IJOB (see above).

RDSUM Input and output parameter.

RDSUM is DOUBLE PRECISION

On entry, the sum of squares of computed contributions to the Dif-estimate under computation by DTGSYL, where the scaling factor RDSCAL (see below) has been factored out. On exit, the corresponding sum of squares updated with the contributions from the current sub-system. If TRANS = 'T' RDSUM is not touched. NOTE: RDSUM only makes sense when DTGSY2 is called by STGSYL.

RDSCAL Input and output parameter.

RDSCAL is DOUBLE PRECISION

On entry, scaling factor used to prevent overflow in RDSUM. On exit, RDSCAL is updated w.r.t. the current contributions in RDSUM. If TRANS = 'T', RDSCAL is not touched. NOTE: RDSCAL only makes sense when DTGSY2 is called by DTGSYL.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N).

The pivot indices; for 1 <= i <= N, row i of the matrix has been interchanged with row IPIV(i).

JPIV Input parameter.

JPIV is INTEGER array, dimension (N).

The pivot indices; for 1 <= j <= N, column j of the matrix has been interchanged with column JPIV(j).

Related Information

For this routine in other precisions, please see *clatdf*, *slatdf* and *zlatdf*.

4.17.360 dlatps

dlatps solves one of the triangular systems

```
A *x = s*b  or  A**T*x = s*b
```

with scaling to prevent overflow, where A is an upper or lower triangular matrix stored in packed form. Here A^T denotes the transpose of A, x and b are n-element vectors, and s is a scaling factor, usually less than or equal to 1, chosen so that the components of x will be less than the overflow threshold. If the unscaled problem will not cause overflow, the Level 2 BLAS routine DTPSV is called. If the matrix A is singular (A(j,j) = 0 for some j), then s is set to 0 and a non-trivial solution to A*x = 0 is returned.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlatps(UPLO, TRANS, DIAG, NORMIN, N, AP, X, SCALE, CNORM, INFO)
```

C specification:

```
#include "armpl.h"

void dlatps_(const char *uplo, const char *trans, const char *diag,
             const char *normin, const armpl_int_t *n, const double *ap,
             double *x, double *scale, double *cnorm, armpl_int_t *info,
             ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the matrix A is upper or lower triangular. = 'U': Upper triangular = 'L': Lower triangular

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the operation applied to A. = 'N': Solve A * x = s*b (No transpose) = 'T': Solve A^T * x = s*b (Transpose) = 'C': Solve A^T * x = s*b (Conjugate transpose = Transpose)

DIAG Input parameter.

DIAG is CHARACTER*1

Specifies whether or not the matrix A is unit triangular. = 'N': Non-unit triangular = 'U': Unit triangular

NORMIN Input parameter.

NORMIN is CHARACTER*1

Specifies whether CNORM has been set or not. = 'Y': CNORM contains the column norms on entry = 'N': CNORM is not set on entry. On exit, the norms will be computed and stored in CNORM.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input parameter.

AP is DOUBLE PRECISION

AP is an array, dimension (N*(N+1)/2). The upper or lower triangular matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

X Input and output parameter.

X is DOUBLE PRECISION

X is an array, dimension (N). On entry, the right hand side b of the triangular system. On exit, X is overwritten by the solution vector x.

SCALE Output parameter.

SCALE is DOUBLE PRECISION

The scaling factor s for the triangular system A * x = s*b or A^T * x = s*b. If SCALE = 0, the matrix A is singular or badly scaled, and the vector x is an exact or approximate solution to A*x = 0.

CNORM Input and output parameter.

CNORM is DOUBLE PRECISION

CNORM is an array, dimension (N). If NORMIN = 'Y', CNORM is an input argument and CNORM(j) contains the norm of the off-diagonal part of the j-th column of A. If TRANS = 'N', CNORM(j) must be greater than or equal to the infinity-norm, and if TRANS = 'T' or 'C', CNORM(j) must be greater than or equal to the 1-norm.

If NORMIN = 'N', CNORM is an output argument and CNORM(j) returns the 1-norm of the offdiagonal part of the j-th column of A.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value

Related Information

For this routine in other precisions, please see *clatps*, *slatps* and *zlatps*.

4.17.361 dlatrd

dlatrd reduces NB rows and columns of a real symmetric matrix A to symmetric tridiagonal form by an orthogonal similarity transformation Q^T * A * Q, and returns the matrices V and W which are needed to apply the transformation to the unreduced part of A.

If UPLO = 'U', dlatrd reduces the last NB rows and columns of a matrix, of which the upper triangle is supplied; if UPLO = 'L', dlatrd reduces the first NB rows and columns of a matrix, of which the lower triangle is supplied.

This is an auxiliary routine called by DSYTRD.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlatrd(UPLO, N, NB, A, LDA, E, TAU, W, LDW)
```

C specification:

```
#include "armpl.h"

void dlatrd_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nb,
             double *a, const armpl_int_t *lda, double *e, double *tau,
             double *w, const armpl_int_t *ldw, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A.

NB Input parameter.

NB is INTEGER

The number of rows and columns to be reduced.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit: if UPLO = 'U', the last NB columns have been reduced to tridiagonal form, with the diagonal elements overwriting the diagonal elements of A; the elements above the diagonal with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors; if UPLO = 'L', the first NB columns have been reduced to tridiagonal form, with the diagonal elements overwriting the diagonal elements of A; the elements below the diagonal with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= (1, N).

E Output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N-1). If UPLO = 'U', E(n-nb:n-1) contains the superdiagonal elements of the last NB columns of the reduced matrix; if UPLO = 'L', E(1:nb) contains the subdiagonal elements of the first NB columns of the reduced matrix.

TAU Output parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (N-1). The scalar factors of the elementary reflectors, stored in TAU(n-nb:n-1) if UPLO = 'U', and in TAU(1:nb) if UPLO = 'L'. See Further Details.

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (LDW, NB). The n-by-nb matrix W required to update the unreduced part of A.

LDW Input parameter.

LDW is INTEGER

The leading dimension of the array W. LDW >= max(1, N).

Related Information

For this routine in other precisions, please see *clatrd*, *slatrd* and *zlatrd*.

4.17.362 dlatrs

`dlatrs` solves one of the triangular systems

```
A *x = s*b  or  A**T *x = s*b
```

with scaling to prevent overflow. Here A is an upper or lower triangular matrix, A^T denotes the transpose of A, x and b are n-element vectors, and s is a scaling factor, usually less than or equal to 1, chosen so that the components of x will be less than the overflow threshold. If the unscaled problem will not cause overflow, the Level 2 BLAS routine DTRSV is called. If the matrix A is singular (A(j,j) = 0 for some j), then s is set to 0 and a non-trivial solution to A*x = 0 is returned.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlatrs(UPLO, TRANS, DIAG, NORMIN, N, A, LDA, X, SCALE, CNORM,
                  INFO)
```

C specification:

```
#include "armpl.h"

void dlatrs_(const char *uplo, const char *trans, const char *diag,
             const char *normin, const armpl_int_t *n, const double *a,
             const armpl_int_t *lda, double *x, double *scale, double *cnorm,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the matrix A is upper or lower triangular. = 'U': Upper triangular = 'L': Lower triangular

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the operation applied to A. = 'N': Solve A * x = s*b (No transpose) = 'T': Solve A^T * x = s*b (Transpose) = 'C': Solve A^T * x = s*b (Conjugate transpose = Transpose)

DIAG Input parameter.

DIAG is CHARACTER*1

Specifies whether or not the matrix A is unit triangular. = 'N': Non-unit triangular = 'U': Unit triangular

NORMIN Input parameter.

NORMIN is CHARACTER*1

Specifies whether CNORM has been set or not. = 'Y': CNORM contains the column norms on entry = 'N': CNORM is not set on entry. On exit, the norms will be computed and stored in CNORM.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). The triangular matrix A. If UPLO = 'U', the leading n by n upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n by n lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max (1, N).

X Input and output parameter.

X is DOUBLE PRECISION

X is an array, dimension (N). On entry, the right hand side b of the triangular system. On exit, X is overwritten by the solution vector x.

SCALE Output parameter.

SCALE is DOUBLE PRECISION

The scaling factor s for the triangular system A * x = s*b or A^T * x = s*b. If SCALE = 0, the matrix A is singular or badly scaled, and the vector x is an exact or approximate solution to A*x = 0.

CNORM Input and output parameter.

CNORM is DOUBLE PRECISION

CNORM is an array, dimension (N). If NORMIN = 'Y', CNORM is an input argument and CNORM(j) contains the norm of the off-diagonal part of the j-th column of A. If TRANS = 'N', CNORM(j) must be greater than or equal to the infinity-norm, and if TRANS = 'T' or 'C', CNORM(j) must be greater than or equal to the 1-norm.

If NORMIN = 'N', CNORM is an output argument and CNORM(j) returns the 1-norm of the offdiagonal part of the j-th column of A.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value

Related Information

For this routine in other precisions, please see *clatrs*, *slatrs* and *zlatrs*.

4.17.363 dlatrz

dlatrz factors the M-by-(M+L) real upper trapezoidal matrix [A1 A2] = [A(1:M,1:M) A(1:M,N-L+1:N)] as (R 0) * Z, by means of orthogonal transformations. Z is an (M+L)-by-(M+L) orthogonal matrix and, R and A1 are M-by-M upper triangular matrices.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlatrz(M, N, L, A, LDA, TAU, WORK)
```

C specification:

```
#include "armpl.h"

void dlatrz_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *l,
             double *a, const armpl_int_t *lda, double *tau, double *work);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

L Input parameter.

L is INTEGER

The number of columns of the matrix A containing the meaningful part of the Householder vectors. N-M >= L >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the leading M-by-N upper trapezoidal part of the array A must contain the matrix to be factorized. On exit, the leading M-by-M upper triangular part of A contains the upper triangular matrix R, and elements N-L+1 to N of the first M rows of A, with the array TAU, represent the orthogonal matrix Z as a product of M elementary reflectors.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (M). The scalar factors of the elementary reflectors.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (M) .

Related Information

For this routine in other precisions, please see *clatrz*, *slatrz* and *zlatrz*.

4.17.364 dlatsqr

dlatsqr computes a blocked Tall-Skinny QR factorization of an M-by-N matrix A, where M >= N: A = Q * R .

Syntax

Fortran specification:

```
use armpl_library

subroutine dlatsqr(M, N, MB, NB, A, LDA, T, LDT, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dlatsqr_(const armpl_int_t *m, const armpl_int_t *n,
              const armpl_int_t *mb, const armpl_int_t *nb, double *a,
              const armpl_int_t *lda, double *t, const armpl_int_t *ldt,
              double *work, const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. M >= N >= 0.

MB Input parameter.

MB is INTEGER

The row block size to be used in the blocked QR. MB > N.

NB Input parameter.

NB is INTEGER

The column block size to be used in the blocked QR. N >= NB >= 1.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the N-by-N upper triangular matrix R; the elements below the diagonal represent Q by the columns of blocked V (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

T Output parameter.

T is DOUBLE PRECISION

T is an array,. dimension (LDT, N * Number_of_row_blocks) where Number_of_row_blocks = CEIL((M-N)/(MB-N)) The blocked upper triangular block reflectors stored in compact form as a sequence of upper triangular blocks. See Further Details below.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= NB.

WORK Output parameter.

(workspace) DOUBLE PRECISION

(workspace) is an array, dimension (MAX(1, LWORK)) .

LWORK Input parameter.

The dimension of the array WORK. LWORK >= NB*N.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *clatsqr*, *slatsqr* and *zlatsqr*.

4.17.365 dlauu2

dlauu2 computes the product U * U^T or L^T * L, where the triangular factor U or L is stored in the upper or lower triangular part of the array A.

If UPLO = 'U' or 'u' then the upper triangle of the result is stored, overwriting the factor U in A. If UPLO = 'L' or 'l' then the lower triangle of the result is stored, overwriting the factor L in A.

This is the unblocked form of the algorithm, calling Level 2 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlauu2(UPLO, N, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void dlauu2_(const char *uplo, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the triangular factor stored in the array A is upper or lower triangular: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the triangular factor U or L. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product U * U^T ; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product L^T * L.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value

Related Information

For this routine in other precisions, please see *clauu2*, *slauu2* and *zlauu2*.

4.17.366 dlauum

dlauum computes the product U * U^T or L^T * L, where the triangular factor U or L is stored in the upper or lower triangular part of the array A.

If UPLO = 'U' or 'u' then the upper triangle of the result is stored, overwriting the factor U in A. If UPLO = 'L' or 'l' then the lower triangle of the result is stored, overwriting the factor L in A.

This is the blocked form of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlauum(UPLO, N, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void dlauum_(const char *uplo, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the triangular factor stored in the array A is upper or lower triangular: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the triangular factor U or L. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product U * U^T ; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product L^T * L.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value

Related Information

For this routine in other precisions, please see *clauum*, *slauum* and *zlauum*. It also exists with a native C interface as *LAPACKE_dlauum*.

4.17.367 dorbdb1

dorbdb1 simultaneously bidiagonalizes the blocks of a tall and skinny matrix X with orthonomal columns:

```
                        [ B11 ]
[ X11 ]   [ P1 |    ] [  0  ]
[-----] = [---------] [-----] Q1**T .
[ X21 ]   [    | P2 ] [ B21 ]
                        [  0  ]
```

X11 is P-by-Q, and X21 is (M-P)-by-Q. Q must be no larger than P, M-P, or M-Q. Routines DORBDB2, DORBDB3, and DORBDB4 handle cases in which Q is not the minimum dimension.

The orthogonal matrices P1, P2, and Q1 are P-by-P, (M-P)-by-(M-P), and (M-Q)-by-(M-Q), respectively. They are represented implicitly by Householder vectors.

B11 and B12 are Q-by-Q bidiagonal matrices represented implicitly by angles THETA, PHI.

Syntax

Fortran specification:

```
use armpl_library

subroutine dorbdb1(M, P, Q, X11, LDX11, X21, LDX21, THETA, PHI, TAUP1, TAUP2,
                   TAUQ1, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dorbdb1_(const armpl_int_t *m, const armpl_int_t *p,
              const armpl_int_t *q, double *x11, const armpl_int_t *ldx11,
              double *x21, const armpl_int_t *ldx21, double *theta,
              double *phi, double *taup1, double *taup2, double *tauq1,
              double *work, const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows X11 plus the number of rows in X21.

P Input parameter.

P is INTEGER

The number of rows in X11. 0 <= P <= M.

Q Input parameter.

Q is INTEGER

The number of columns in X11 and X21. 0 <= Q <= MIN(P,M-P,M-Q).

X11 Input and output parameter.

X11 is DOUBLE PRECISION

X11 is an array, dimension (LDX11, Q). On entry, the top block of the matrix X to be reduced. On exit, the columns of tril(X11) specify reflectors for P1 and the rows of triu(X11,1) specify reflectors for Q1.

LDX11 Input parameter.

LDX11 is INTEGER

The leading dimension of X11. LDX11 >= P.

X21 Input and output parameter.

X21 is DOUBLE PRECISION

X21 is an array, dimension (LDX21, Q). On entry, the bottom block of the matrix X to be reduced. On exit, the columns of tril(X21) specify reflectors for P2.

LDX21 Input parameter.

LDX21 is INTEGER

The leading dimension of X21. LDX21 >= M-P.

THETA Output parameter.

THETA is DOUBLE PRECISION

THETA is an array, dimension (Q). The entries of the bidiagonal blocks B11, B21 are defined by THETA and PHI. See Further Details.

PHI Output parameter.

PHI is DOUBLE PRECISION

PHI is an array, dimension (Q-1). The entries of the bidiagonal blocks B11, B21 are defined by THETA and PHI. See Further Details.

TAUP1 Output parameter.

TAUP1 is DOUBLE PRECISION

TAUP1 is an array, dimension (P). The scalar factors of the elementary reflectors that define P1.

TAUP2 Output parameter.

TAUP2 is DOUBLE PRECISION

TAUP2 is an array, dimension (M-P). The scalar factors of the elementary reflectors that define P2.

TAUQ1 Output parameter.

TAUQ1 is DOUBLE PRECISION

TAUQ1 is an array, dimension (Q). The scalar factors of the elementary reflectors that define Q1.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= M-Q.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *sorbdb1*.

4.17.368 dorbdb2

dorbdb2 simultaneously bidiagonalizes the blocks of a tall and skinny matrix X with orthonomal columns:

```
                        [ B11 ]
[ X11 ]   [ P1 |    ] [  0  ]
[-----] = [---------] [-----] Q1**T .
[ X21 ]   [    | P2 ] [ B21 ]
                        [  0  ]
```

X11 is P-by-Q, and X21 is (M-P)-by-Q. P must be no larger than M-P, Q, or M-Q. Routines DORBDB1, DORBDB3, and DORBDB4 handle cases in which P is not the minimum dimension.

The orthogonal matrices P1, P2, and Q1 are P-by-P, (M-P)-by-(M-P), and (M-Q)-by-(M-Q), respectively. They are represented implicitly by Householder vectors.

B11 and B12 are P-by-P bidiagonal matrices represented implicitly by angles THETA, PHI.

Syntax

Fortran specification:

```
use armpl_library

subroutine dorbdb2(M, P, Q, X11, LDX11, X21, LDX21, THETA, PHI, TAUP1, TAUP2,
                   TAUQ1, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dorbdb2_(const armpl_int_t *m, const armpl_int_t *p,
              const armpl_int_t *q, double *x11, const armpl_int_t *ldx11,
              double *x21, const armpl_int_t *ldx21, double *theta,
              double *phi, double *taup1, double *taup2, double *tauq1,
              double *work, const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows X11 plus the number of rows in X21.

P Input parameter.

P is INTEGER

The number of rows in X11. 0 <= P <= min(M-P, Q,M-Q).

Q Input parameter.

Q is INTEGER

The number of columns in X11 and X21. 0 <= Q <= M.

X11 Input and output parameter.

X11 is DOUBLE PRECISION

X11 is an array, dimension (LDX11, Q). On entry, the top block of the matrix X to be reduced. On exit, the columns of tril(X11) specify reflectors for P1 and the rows of triu(X11,1) specify reflectors for Q1.

LDX11 Input parameter.

LDX11 is INTEGER

The leading dimension of X11. LDX11 >= P.

X21 Input and output parameter.

X21 is DOUBLE PRECISION

X21 is an array, dimension (LDX21, Q). On entry, the bottom block of the matrix X to be reduced. On exit, the columns of tril(X21) specify reflectors for P2.

LDX21 Input parameter.

LDX21 is INTEGER

The leading dimension of X21. LDX21 >= M-P.

THETA Output parameter.

THETA is DOUBLE PRECISION

THETA is an array, dimension (Q). The entries of the bidiagonal blocks B11, B21 are defined by THETA and PHI. See Further Details.

PHI Output parameter.

PHI is DOUBLE PRECISION

PHI is an array, dimension (Q-1). The entries of the bidiagonal blocks B11, B21 are defined by THETA and PHI. See Further Details.

TAUP1 Output parameter.

TAUP1 is DOUBLE PRECISION

TAUP1 is an array, dimension (P). The scalar factors of the elementary reflectors that define P1.

TAUP2 Output parameter.

TAUP2 is DOUBLE PRECISION

TAUP2 is an array, dimension (M-P). The scalar factors of the elementary reflectors that define P2.

TAUQ1 Output parameter.

TAUQ1 is DOUBLE PRECISION

TAUQ1 is an array, dimension (Q). The scalar factors of the elementary reflectors that define Q1.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= M-Q.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *sorbdb2*.

4.17.369 dorbdb3

dorbdb3 simultaneously bidiagonalizes the blocks of a tall and skinny matrix X with orthonomal columns:

```
                        [ B11 ]
[ X11 ]   [ P1 |    ] [  0  ]
[-----] = [---------] [-----] Q1**T .
[ X21 ]   [    | P2 ] [ B21 ]
                        [  0  ]
```

X11 is P-by-Q, and X21 is (M-P)-by-Q. M-P must be no larger than P, Q, or M-Q. Routines DORBDB1, DORBDB2, and DORBDB4 handle cases in which M-P is not the minimum dimension.

The orthogonal matrices P1, P2, and Q1 are P-by-P, (M-P)-by-(M-P), and (M-Q)-by-(M-Q), respectively. They are represented implicitly by Householder vectors.

B11 and B12 are (M-P)-by-(M-P) bidiagonal matrices represented implicitly by angles THETA, PHI.

Syntax

Fortran specification:

```
use armpl_library

subroutine dorbdb3(M, P, Q, X11, LDX11, X21, LDX21, THETA, PHI, TAUP1, TAUP2,
                   TAUQ1, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dorbdb3_(const armpl_int_t *m, const armpl_int_t *p,
              const armpl_int_t *q, double *x11, const armpl_int_t *ldx11,
              double *x21, const armpl_int_t *ldx21, double *theta,
              double *phi, double *taup1, double *taup2, double *tauq1,
              double *work, const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows X11 plus the number of rows in X21.

P Input parameter.

P is INTEGER

The number of rows in X11. 0 <= P <= M. M-P <= min(P, Q,M-Q).

Q Input parameter.

Q is INTEGER

The number of columns in X11 and X21. 0 <= Q <= M.

X11 Input and output parameter.

X11 is DOUBLE PRECISION

X11 is an array, dimension (LDX11, Q). On entry, the top block of the matrix X to be reduced. On exit, the columns of tril(X11) specify reflectors for P1 and the rows of triu(X11,1) specify reflectors for Q1.

LDX11 Input parameter.

LDX11 is INTEGER

The leading dimension of X11. LDX11 >= P.

X21 Input and output parameter.

X21 is DOUBLE PRECISION

X21 is an array, dimension (LDX21, Q). On entry, the bottom block of the matrix X to be reduced. On exit, the columns of tril(X21) specify reflectors for P2.

LDX21 Input parameter.

LDX21 is INTEGER

The leading dimension of X21. LDX21 >= M-P.

THETA Output parameter.

THETA is DOUBLE PRECISION

THETA is an array, dimension (Q). The entries of the bidiagonal blocks B11, B21 are defined by THETA and PHI. See Further Details.

PHI Output parameter.

PHI is DOUBLE PRECISION

PHI is an array, dimension (Q-1). The entries of the bidiagonal blocks B11, B21 are defined by THETA and PHI. See Further Details.

TAUP1 Output parameter.

TAUP1 is DOUBLE PRECISION

TAUP1 is an array, dimension (P). The scalar factors of the elementary reflectors that define P1.

TAUP2 Output parameter.

TAUP2 is DOUBLE PRECISION

TAUP2 is an array, dimension (M-P). The scalar factors of the elementary reflectors that define P2.

TAUQ1 Output parameter.

TAUQ1 is DOUBLE PRECISION

TAUQ1 is an array, dimension (Q). The scalar factors of the elementary reflectors that define Q1.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= M-Q.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *sorbdb3*.

4.17.370 dorbdb4

dorbdb4 simultaneously bidiagonalizes the blocks of a tall and skinny matrix X with orthonomal columns:

```
                         [ B11 ]
[ X11 ]   [ P1 |    ] [  0  ]
[-----] = [---------] [-----] Q1**T .
[ X21 ]   [    | P2 ] [ B21 ]
                         [  0  ]
```

X11 is P-by-Q, and X21 is (M-P)-by-Q. M-Q must be no larger than P, M-P, or Q. Routines DORBDB1, DORBDB2, and DORBDB3 handle cases in which M-Q is not the minimum dimension.

The orthogonal matrices P1, P2, and Q1 are P-by-P, (M-P)-by-(M-P), and (M-Q)-by-(M-Q), respectively. They are represented implicitly by Householder vectors.

B11 and B12 are (M-Q)-by-(M-Q) bidiagonal matrices represented implicitly by angles THETA, PHI.

Syntax

Fortran specification:

```
use armpl_library

subroutine dorbdb4(M, P, Q, X11, LDX11, X21, LDX21, THETA, PHI, TAUP1, TAUP2,
                   TAUQ1, PHANTOM, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dorbdb4_(const armpl_int_t *m, const armpl_int_t *p,
              const armpl_int_t *q, double *x11, const armpl_int_t *ldx11,
              double *x21, const armpl_int_t *ldx21, double *theta,
              double *phi, double *taup1, double *taup2, double *tauq1,
              double *phantom, double *work, const armpl_int_t *lwork,
              armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows X11 plus the number of rows in X21.

P Input parameter.

P is INTEGER

The number of rows in X11. 0 <= P <= M.

Q Input parameter.

Q is INTEGER

The number of columns in X11 and X21. 0 <= Q <= M and M-Q <= min(P,M-P, Q).

X11 Input and output parameter.

X11 is DOUBLE PRECISION

X11 is an array, dimension (LDX11, Q). On entry, the top block of the matrix X to be reduced. On exit, the columns of tril(X11) specify reflectors for P1 and the rows of triu(X11,1) specify reflectors for Q1.

LDX11 Input parameter.

LDX11 is INTEGER

The leading dimension of X11. LDX11 >= P.

X21 Input and output parameter.

X21 is DOUBLE PRECISION

X21 is an array, dimension (LDX21, Q). On entry, the bottom block of the matrix X to be reduced. On exit, the columns of tril(X21) specify reflectors for P2.

LDX21 Input parameter.

LDX21 is INTEGER

The leading dimension of X21. LDX21 >= M-P.

THETA Output parameter.

THETA is DOUBLE PRECISION

THETA is an array, dimension (Q). The entries of the bidiagonal blocks B11, B21 are defined by THETA and PHI. See Further Details.

PHI Output parameter.

PHI is DOUBLE PRECISION

PHI is an array, dimension (Q-1). The entries of the bidiagonal blocks B11, B21 are defined by THETA and PHI. See Further Details.

TAUP1 Output parameter.

TAUP1 is DOUBLE PRECISION

TAUP1 is an array, dimension (P). The scalar factors of the elementary reflectors that define P1.

TAUP2 Output parameter.

TAUP2 is DOUBLE PRECISION

TAUP2 is an array, dimension (M-P). The scalar factors of the elementary reflectors that define P2.

TAUQ1 Output parameter.

TAUQ1 is DOUBLE PRECISION

TAUQ1 is an array, dimension (Q). The scalar factors of the elementary reflectors that define Q1.

PHANTOM Output parameter.

PHANTOM is DOUBLE PRECISION

PHANTOM is an array, dimension (M). The routine computes an M-by-1 column vector Y that is orthogonal to the columns of [X11; X21]. PHANTOM(1:P) and PHANTOM(P+1:M) contain Householder vectors for Y(1:P) and Y(P+1:M), respectively.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= M-Q.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *sorbdb4*.

4.17.371 dorbdb5

`dorbdb5` orthogonalizes the column vector

```
X = [ X1 ]
    [ X2 ]
```

with respect to the columns of

```
Q = [ Q1 ] .
    [ Q2 ]
```

The columns of Q must be orthonormal.

If the projection is zero according to Kahan's "twice is enough" criterion, then some other vector from the orthogonal complement is returned. This vector is chosen in an arbitrary but deterministic way.

Syntax

Fortran specification:

```
use armpl_library

subroutine dorbdb5(M1, M2, N, X1, INCX1, X2, INCX2, Q1, LDQ1, Q2, LDQ2, WORK,
                   LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dorbdb5_(const armpl_int_t *m1, const armpl_int_t *m2,
              const armpl_int_t *n, double *x1, const armpl_int_t *incx1,
              double *x2, const armpl_int_t *incx2, double *q1,
              const armpl_int_t *ldq1, double *q2, const armpl_int_t *ldq2,
              double *work, const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M1 Input parameter.

`M1` is `INTEGER`

The dimension of `X1` and the number of rows in `Q1`. 0 <= `M1`.

M2 Input parameter.

`M2` is `INTEGER`

The dimension of `X2` and the number of rows in `Q2`. 0 <= `M2`.

N Input parameter.

`N` is `INTEGER`

The number of columns in `Q1` and `Q2`. 0 <= `N`.

X1 Input and output parameter.

`X1` is `DOUBLE PRECISION`

`X1` is an array, dimension (`M1`). On entry, the top part of the vector to be orthogonalized. On exit, the top part of the projected vector.

INCX1 Input parameter.

INCX1 is INTEGER

Increment for entries of X1.

X2 Input and output parameter.

X2 is DOUBLE PRECISION

X2 is an array, dimension (M2). On entry, the bottom part of the vector to be orthogonalized. On exit, the bottom part of the projected vector.

INCX2 Input parameter.

INCX2 is INTEGER

Increment for entries of X2.

Q1 Input parameter.

Q1 is DOUBLE PRECISION

Q1 is an array, dimension (LDQ1, N). The top part of the orthonormal basis matrix.

LDQ1 Input parameter.

LDQ1 is INTEGER

The leading dimension of Q1. LDQ1 >= M1.

Q2 Input parameter.

Q2 is DOUBLE PRECISION

Q2 is an array, dimension (LDQ2, N). The bottom part of the orthonormal basis matrix.

LDQ2 Input parameter.

LDQ2 is INTEGER

The leading dimension of Q2. LDQ2 >= M2.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= N.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *sorbdb5*.

4.17.372 dorbdb6

dorbdb6 orthogonalizes the column vector

```
X = [ X1 ]
    [ X2 ]
```

with respect to the columns of

```
Q = [ Q1 ] .
    [ Q2 ]
```

The columns of Q must be orthonormal.

If the projection is zero according to Kahan's "twice is enough" criterion, then the zero vector is returned.

Syntax

Fortran specification:

```
use armpl_library

subroutine dorbdb6(M1, M2, N, X1, INCX1, X2, INCX2, Q1, LDQ1, Q2, LDQ2, WORK,
                   LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dorbdb6_(const armpl_int_t *m1, const armpl_int_t *m2,
              const armpl_int_t *n, double *x1, const armpl_int_t *incx1,
              double *x2, const armpl_int_t *incx2, double *q1,
              const armpl_int_t *ldq1, double *q2, const armpl_int_t *ldq2,
              double *work, const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M1 Input parameter.

M1 is INTEGER

The dimension of X1 and the number of rows in Q1. 0 <= M1.

M2 Input parameter.

M2 is INTEGER

The dimension of X2 and the number of rows in Q2. 0 <= M2.

N Input parameter.

N is INTEGER

The number of columns in Q1 and Q2. 0 <= N.

X1 Input and output parameter.

X1 is DOUBLE PRECISION

X1 is an array, dimension (M1). On entry, the top part of the vector to be orthogonalized. On exit, the top part of the projected vector.

INCX1 Input parameter.

INCX1 is INTEGER

Increment for entries of X1.

X2 Input and output parameter.

X2 is DOUBLE PRECISION

X2 is an array, dimension (M2). On entry, the bottom part of the vector to be orthogonalized. On exit, the bottom part of the projected vector.

INCX2 Input parameter.

INCX2 is INTEGER

Increment for entries of X2.

Q1 Input parameter.

Q1 is DOUBLE PRECISION

Q1 is an array, dimension (LDQ1, N). The top part of the orthonormal basis matrix.

LDQ1 Input parameter.

LDQ1 is INTEGER

The leading dimension of Q1. LDQ1 >= M1.

Q2 Input parameter.

Q2 is DOUBLE PRECISION

Q2 is an array, dimension (LDQ2, N). The bottom part of the orthonormal basis matrix.

LDQ2 Input parameter.

LDQ2 is INTEGER

The leading dimension of Q2. LDQ2 >= M2.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= N.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *sorbdb6*.

4.17.373 dorg2l

dorg2l generates an m by n real matrix Q with orthonormal columns, which is defined as the last n columns of a product of k elementary reflectors of order m

```
Q  =  H(k) . . . H(2) H(1)
```

as returned by DGEQLF.

Syntax

Fortran specification:

```
use armpl_library

subroutine dorg2l(M, N, K, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dorg2l_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *k,
             double *a, const armpl_int_t *lda, const double *tau,
             double *work, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix Q. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix Q. M >= N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. N >= K >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the (n-k+i)-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by DGEQLF in the last k columns of its array argument A. On exit, the m by n matrix Q.

LDA Input parameter.

LDA is INTEGER

The first dimension of the array A. LDA >= max(1, M).

TAU Input parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by DGEQLF.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument has an illegal value

Related Information

For this routine in other precisions, please see *sorg2l*.

4.17.374 dorg2r

dorg2r generates an m by n real matrix Q with orthonormal columns, which is defined as the first n columns of a product of k elementary reflectors of order m

```
Q  =  H(1) H(2) . . . H(k)
```

as returned by DGEQRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine dorg2r(M, N, K, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dorg2r_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *k,
             double *a, const armpl_int_t *lda, const double *tau,
             double *work, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix Q. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix Q. M >= N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. N >= K >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the i-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by DGEQRF in the first k columns of its array argument A. On exit, the m-by-n matrix Q.

LDA Input parameter.

LDA is INTEGER

The first dimension of the array A. LDA >= max(1, M).

TAU Input parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by DGEQRF.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument has an illegal value

Related Information

For this routine in other precisions, please see *sorg2r*.

4.17.375 dorgl2

dorgl2 generates an m by n real matrix Q with orthonormal rows, which is defined as the first m rows of a product of k elementary reflectors of order n

```
Q  =  H(k) . . . H(2) H(1)
```

as returned by DGELQF.

Syntax

Fortran specification:

```
use armpl_library

subroutine dorgl2(M, N, K, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dorgl2_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *k,
             double *a, const armpl_int_t *lda, const double *tau,
             double *work, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix Q. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix Q. N >= M.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. M >= K >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,. . . ,k, as returned by DGELQF in the first k rows of its array argument A. On exit, the m-by-n matrix Q.

LDA Input parameter.

LDA is INTEGER

The first dimension of the array A. LDA >= max(1, M).

TAU Input parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by DGELQF.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (M) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument has an illegal value

Related Information

For this routine in other precisions, please see *sorgl2*.

4.17.376 dorgr2

dorgr2 generates an m by n real matrix Q with orthonormal rows, which is defined as the last m rows of a product of k elementary reflectors of order n

```
Q  =  H(1) H(2) . . . H(k)
```

as returned by DGERQF.

Syntax

Fortran specification:

```
use armpl_library

subroutine dorgr2(M, N, K, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dorgr2_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *k,
             double *a, const armpl_int_t *lda, const double *tau,
             double *work, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix Q. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix Q. N >= M.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. M >= K >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the (m-k+i)-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by DGERQF in the last k rows of its array argument A. On exit, the m by n matrix Q.

LDA Input parameter.

LDA is INTEGER

The first dimension of the array A. LDA >= max(1, M).

TAU Input parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by DGERQF.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (M) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument has an illegal value

Related Information

For this routine in other precisions, please see *sorgr2*.

4.17.377 dorm2l

dorm2l overwrites the general real m by n matrix C with

```
Q * C  if SIDE = 'L' and TRANS = 'N', or

Q**T * C  if SIDE = 'L' and TRANS = 'T', or

C * Q  if SIDE = 'R' and TRANS = 'N', or

C * Q**T if SIDE = 'R' and TRANS = 'T',
```

where Q is a real orthogonal matrix defined as the product of k elementary reflectors

```
Q = H(k) . . . H(2) H(1)
```

as returned by DGEQLF. Q is of order m if SIDE = 'L' and of order n if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine dorm2l(SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dorm2l_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k, const double *a,
             const armpl_int_t *lda, const double *tau, double *c,
             const armpl_int_t *ldc, double *work, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left = 'R': apply Q or Q^T from the Right

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply Q (No transpose) = 'T': apply Q^T (Transpose)

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, K). The i-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by DGEQLF in the last k columns of its array argument A. A is modified by the routine but restored on exit.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDA >= max(1, M); if SIDE = 'R', LDA >= max(1, N).

TAU Input parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by DGEQLF.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, N). On entry, the m by n matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension. (N) if SIDE = 'L', (M) if SIDE = 'R'

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *sorm2l*.

4.17.378 dorm2r

dorm2r overwrites the general real m by n matrix C with

```
Q * C  if SIDE = 'L' and TRANS = 'N', or

Q**T* C  if SIDE = 'L' and TRANS = 'T', or

C * Q  if SIDE = 'R' and TRANS = 'N', or

C * Q**T if SIDE = 'R' and TRANS = 'T',
```

where Q is a real orthogonal matrix defined as the product of k elementary reflectors

```
Q = H(1) H(2) . . . H(k)
```

as returned by DGEQRF. Q is of order m if SIDE = 'L' and of order n if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine dorm2r(SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dorm2r_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k, const double *a,
             const armpl_int_t *lda, const double *tau, double *c,
             const armpl_int_t *ldc, double *work, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left = 'R': apply Q or Q^T from the Right

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply Q (No transpose) = 'T': apply Q^T (Transpose)

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, K). The i-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by DGEQRF in the first k columns of its array argument A. A is modified by the routine but restored on exit.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDA >= max(1, M); if SIDE = 'R', LDA >= max(1, N).

TAU Input parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by DGEQRF.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, N). On entry, the m by n matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension. (N) if SIDE = 'L', (M) if SIDE = 'R'

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *sorm2r*.

4.17.379 dorml2

dorml2 overwrites the general real m by n matrix C with

```
Q * C  if SIDE = 'L' and TRANS = 'N', or

Q**T* C  if SIDE = 'L' and TRANS = 'T', or

C * Q  if SIDE = 'R' and TRANS = 'N', or

C * Q**T if SIDE = 'R' and TRANS = 'T',
```

where Q is a real orthogonal matrix defined as the product of k elementary reflectors

```
Q = H(k) . . . H(2) H(1)
```

as returned by DGELQF. Q is of order m if SIDE = 'L' and of order n if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine dorml2(SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dorml2_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k, const double *a,
             const armpl_int_t *lda, const double *tau, double *c,
             const armpl_int_t *ldc, double *work, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left = 'R': apply Q or Q^T from the Right

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply Q (No transpose) = 'T': apply Q^T (Transpose)

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension. (LDA, M) if SIDE = 'L', (LDA, N) if SIDE = 'R' The i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by DGELQF in the first k rows of its array argument A. A is modified by the routine but restored on exit.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, K).

TAU Input parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by DGELQF.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, N). On entry, the m by n matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension. (N) if SIDE = 'L', (M) if SIDE = 'R'

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *sorml2*.

4.17.380 dormr2

dormr2 overwrites the general real m by n matrix C with

```
Q * C  if SIDE = 'L' and TRANS = 'N', or

Q**T* C  if SIDE = 'L' and TRANS = 'T', or

C * Q  if SIDE = 'R' and TRANS = 'N', or

C * Q**T if SIDE = 'R' and TRANS = 'T',
```

where Q is a real orthogonal matrix defined as the product of k elementary reflectors

```
Q = H(1) H(2) . . . H(k)
```

as returned by DGERQF. Q is of order m if SIDE = 'L' and of order n if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine dormr2(SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dormr2_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k, const double *a,
             const armpl_int_t *lda, const double *tau, double *c,
             const armpl_int_t *ldc, double *work, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left = 'R': apply Q or Q^T from the Right

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply Q (No transpose) = 'T': apply Q' (Transpose)

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension. (LDA, M) if SIDE = 'L', (LDA, N) if SIDE = 'R' The i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by DGERQF in the last k rows of its array argument A. A is modified by the routine but restored on exit.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, K).

TAU Input parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by DGERQF.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, N). On entry, the m by n matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension. (N) if SIDE = 'L', (M) if SIDE = 'R'

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *sormr2*.

4.17.381 dormr3

dormr3 overwrites the general real m by n matrix C with

```
Q * C  if SIDE = 'L' and TRANS = 'N', or

Q**T* C  if SIDE = 'L' and TRANS = 'C', or

C * Q  if SIDE = 'R' and TRANS = 'N', or

C * Q**T if SIDE = 'R' and TRANS = 'C',
```

where Q is a real orthogonal matrix defined as the product of k elementary reflectors

```
Q = H(1) H(2) . . . H(k)
```

as returned by DTZRZF. Q is of order m if SIDE = 'L' and of order n if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine dormr3(SIDE, TRANS, M, N, K, L, A, LDA, TAU, C, LDC, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dormr3_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k, const armpl_int_t *l,
             const double *a, const armpl_int_t *lda, const double *tau,
             double *c, const armpl_int_t *ldc, double *work,
             armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left = 'R': apply Q or Q^T from the Right

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply Q (No transpose) = 'T': apply Q^T (Transpose)

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

L Input parameter.

L is INTEGER

The number of columns of the matrix A containing the meaningful part of the Householder reflectors. If SIDE = 'L', M >= L >= 0, if SIDE = 'R', N >= L >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension. (LDA, M) if SIDE = 'L', (LDA, N) if SIDE = 'R' The i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by DTZRZF in the last k rows of its array argument A. A is modified by the routine but restored on exit.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, K).

TAU Input parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by DTZRZF.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, N). On entry, the m-by-n matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension. (N) if SIDE = 'L', (M) if SIDE = 'R'

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *sormr3*.

4.17.382 dpbtf2

dpbtf2 computes the Cholesky factorization of a real symmetric positive definite band matrix A.

The factorization has the form

```
A = U**T * U ,  if UPLO = 'U', or
A = L  * L**T,  if UPLO = 'L',
```

where U is an upper triangular matrix, U^T is the transpose of U, and L is lower triangular.

This is the unblocked version of the algorithm, calling Level 2 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine dpbtf2(UPLO, N, KD, AB, LDAB, INFO)
```

C specification:

```
#include "armpl.h"

void dpbtf2_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
             double *ab, const armpl_int_t *ldab, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of super-diagonals of the matrix A if UPLO = 'U', or the number of sub-diagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is DOUBLE PRECISION

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, if INFO = 0, the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T of the band matrix A, in the same storage format as A.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value > 0: if INFO = k, the leading minor of order k is not positive definite, and the factorization could not be completed.

Related Information

For this routine in other precisions, please see *cpbtf2*, *spbtf2* and *zpbtf2*.

4.17.383 dpotf2

dpotf2 computes the Cholesky factorization of a real symmetric positive definite matrix A.

The factorization has the form

```
A = U**T * U ,  if UPLO = 'U', or
A = L  * L**T,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is lower triangular.

This is the unblocked version of the algorithm, calling Level 2 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine dpotf2(UPLO, N, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void dpotf2_(const char *uplo, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored. = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading n by n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n by n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T .

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value > 0: if INFO = k, the leading minor of order k is not positive definite, and the factorization could not be completed.

Related Information

For this routine in other precisions, please see *cpotf2*, *spotf2* and *zpotf2*.

4.17.384 dpstf2

dpstf2 computes the Cholesky factorization with complete pivoting of a real symmetric positive semidefinite matrix A.

The factorization has the form

```
P**T * A * P = U**T * U ,  if UPLO = 'U',
P**T * A * P = L  * L**T,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is lower triangular, and P is stored as vector PIV.

This algorithm does not attempt to check that A is positive semidefinite. This version of the algorithm calls level 2 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine dpstf2(UPLO, N, A, LDA, PIV, RANK, TOL, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void dpstf2_(const char *uplo, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, armpl_int_t *piv, armpl_int_t *rank,
             const double *tol, double *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored. = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading n by n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n by n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization as above.

PIV Output parameter.

PIV is INTEGER array, dimension (N)

PIV is such that the nonzero entries are P(PIV(K), K) = 1.

RANK Output parameter.

RANK is INTEGER

The rank of A given by the number of steps the algorithm completed.

TOL Input parameter.

TOL is DOUBLE PRECISION

User defined tolerance. If TOL < 0, then N*U*MAX(A(K,K)) will be used. The algorithm terminates at the (K-1)st step if the pivot <= TOL.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (2*N). Work space.

INFO Output parameter.

INFO is INTEGER

< 0: If INFO = -K, the K-th argument had an illegal value, = 0: algorithm completed successfully, and > 0: the matrix A is either rank deficient with computed rank as returned in RANK, or is not positive semidefinite. See Section 7 of LAPACK Working Note #161 for further information.

Related Information

For this routine in other precisions, please see *cpstf2*, *spstf2* and *zpstf2*.

4.17.385 dptts2

dptts2 solves a tridiagonal system of the form

```
A * X = B
```

using the L*D*L^T factorization of A computed by DPTTRF. D is a diagonal matrix specified in the vector D, L is a unit bidiagonal matrix whose subdiagonal is specified in the vector E, and X and B are N by NRHS matrices.

Syntax

Fortran specification:

```
use armpl_library

subroutine dptts2(N, NRHS, D, E, B, LDB)
```

C specification:

```
#include "armpl.h"

void dptts2_(const armpl_int_t *n, const armpl_int_t *nrhs, const double *d,
             const double *e, double *b, const armpl_int_t *ldb);
```

Parameters

N Input parameter.

N is INTEGER

The order of the tridiagonal matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The n diagonal elements of the diagonal matrix D from the L*D*L^T factorization of A.

E Input parameter.

E is DOUBLE PRECISION

E is an array, dimension (N-1). The (n-1) subdiagonal elements of the unit bidiagonal factor L from the L*D*L^T factorization of A. E can also be regarded as the superdiagonal of the unit bidiagonal factor U from the factorization A = U^T *D*U.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, NRHS). On entry, the right hand side vectors B for the system of linear equations. On exit, the solution vectors, X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Related Information

For this routine in other precisions, please see *cptts2*, *sptts2* and *zptts2*.

4.17.386 drscl

drscl multiplies an n-element real vector x by the real scalar 1/a. This is done without overflow or underflow as long as the final result x/a does not overflow or underflow.

Syntax

Fortran specification:

```
use armpl_library

subroutine drscl(N, SA, SX, INCX)
```

C specification:

```
#include "armpl.h"

void drscl_(const armpl_int_t *n, const double *sa, double *sx,
            const armpl_int_t *incx);
```

Parameters

N Input parameter.

N is INTEGER

The number of components of the vector x.

SA Input parameter.

SA is DOUBLE PRECISION

The scalar a which is used to divide each component of x. SA must be >= 0, or the subroutine will divide by zero.

SX Input and output parameter.

SX is DOUBLE PRECISION

SX is an array, dimension. (1+(N-1)*abs(INCX)) The n-element vector x.

INCX Input parameter.

INCX is INTEGER

The increment between successive values of the vector SX. > 0: SX(1) = X(1) and SX(1+(i-1)*INCX) = x(i), 1< i<= n

Related Information

For this routine in other precisions, please see *srscl*.

4.17.387 dsfrk

Level 3 BLAS like routine for C in RFP Format.

dsfrk performs one of the symmetric rank–k operations

```
C := alpha*A*A**T + beta*C,
```

or

```
C := alpha*A**T*A + beta*C,
```

where alpha and beta are real scalars, C is an n–by–n symmetric matrix and A is an n–by–k matrix in the first case and a k–by–n matrix in the second case.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsfrk(TRANSR, UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C)
```

C specification:

```
#include "armpl.h"

void dsfrk_(const char *transr, const char *uplo, const char *trans,
            const armpl_int_t *n, const armpl_int_t *k, const double *alpha,
            const double *a, const armpl_int_t *lda, const double *beta,
            double *c, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': The Normal Form of RFP A is stored; = 'T': The Transpose Form of RFP A is stored.

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the array C is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of C is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of C is to be referenced.

Unchanged on exit.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' C := alpha*A*A^T + beta*C.

TRANS = 'T' or 't' C := alpha*A^T * A + beta*C.

Unchanged on exit.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix C. N must be at least zero. Unchanged on exit.

K Input parameter.

K is INTEGER

On entry with TRANS = 'N' or 'n', K specifies the number of columns of the matrix A, and on entry with TRANS = 'T' or 't', K specifies the number of rows of the matrix A. K must be at least zero. Unchanged on exit.

ALPHA Input parameter.

ALPHA is DOUBLE PRECISION

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA,ka). where KA is K when TRANS = 'N' or 'n', and is N otherwise. Before entry with TRANS = 'N' or 'n', the leading N–by–K part of the array A must contain the matrix A, otherwise the leading K–by–N part of the array A must contain the matrix A. Unchanged on exit.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDA must be at least max(1, n), otherwise LDA must be at least max(1, k). Unchanged on exit.

BETA Input parameter.

BETA is DOUBLE PRECISION

On entry, BETA specifies the scalar beta. Unchanged on exit.

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (NT). NT = N*(N+1)/2. On entry, the symmetric matrix C in RFP Format. RFP Format is described by TRANSR, UPLO and N.

Related Information

For this routine in other precisions, please see *ssfrk*. It also exists with a native C interface as *LAPACKE_dsfrk*.

4.17.388 dsyconv

dsyconv convert A given by TRF into L and D and vice-versa. Get Non-diag elements of D (returned in workspace) and apply or reverse permutation done in TRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsyconv(UPLO, WAY, N, A, LDA, IPIV, E, INFO)
```

C specification:

```
#include "armpl.h"

void dsyconv_(const char *uplo, const char *way, const armpl_int_t *n,
              double *a, const armpl_int_t *lda, const armpl_int_t *ipiv,
              double *e, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

WAY Input parameter.

WAY is CHARACTER*1

= 'C': Convert = 'R': Revert

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by DSYTRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by DSYTRF.

E Output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N). E stores the supdiagonal/subdiagonal of the symmetric 1-by-1 or 2-by-2 block diagonal matrix D in LDLT.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csyconv*, *ssyconv* and *zsyconv*. It also exists with a native C interface as *LAPACKE_dsyconv*.

4.17.389 dsygs2

dsygs2 reduces a real symmetric-definite generalized eigenproblem to standard form.

If ITYPE = 1, the problem is A*x = lambda*B*x, and A is overwritten by inv(U^T)*A*inv(U) or inv(L)*A*inv(L^T)

If ITYPE = 2 or 3, the problem is A*B*x = lambda*x or B*A*x = lambda*x, and A is overwritten by U*A*U^T or L^T*A*L.

B must have been previously factorized as U^T*U or L*L^T by DPOTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsygs2(ITYPE, UPLO, N, A, LDA, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void dsygs2_(const armpl_int_t *itype, const char *uplo, const armpl_int_t *n,
             double *a, const armpl_int_t *lda, const double *b,
             const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

ITYPE Input parameter.

ITYPE is INTEGER

= 1: compute inv(U^T)*A*inv(U) or inv(L)*A*inv(L^T); = 2 or 3: compute U*A*U^T or L^T*A*L.

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored, and how B has been factorized. = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading n by n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n by n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the transformed matrix, stored in the same format as A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). The triangular factor from the Cholesky factorization of B, as returned by DPOTRF.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *ssygs2*.

4.17.390 dsyswapr

DSYSWAPR applies an elementary permutation on the rows and the columns of a symmetric matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsyswapr(UPLO, N, A, LDA, I1, I2)
```

C specification:

```
#include "armpl.h"

void dsyswapr_(const char *uplo, const armpl_int_t *n, double *a,
               const armpl_int_t *lda, const armpl_int_t *i1,
               const armpl_int_t *i2, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the NB diagonal matrix D and the multipliers used to obtain the factor U or L as computed by DSYTRF.

On exit, if INFO = 0, the (symmetric) inverse of the original matrix. If UPLO = 'U', the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; if UPLO = 'L' the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

I1 Input parameter.

I1 is INTEGER

Index of the first row to swap

I2 Input parameter.

I2 is INTEGER

Index of the second row to swap

Related Information

For this routine in other precisions, please see *csyswapr*, *ssyswapr* and *zsyswapr*. It also exists with a native C interface as *LAPACKE_dsyswapr*.

4.17.391 dsytd2

`dsytd2` reduces a real symmetric matrix A to symmetric tridiagonal form T by an orthogonal similarity transformation: $Q^T * A * Q = T$.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsytd2(UPLO, N, A, LDA, D, E, TAU, INFO)
```

C specification:

```
#include "armpl.h"

void dsytd2_(const char *uplo, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, double *d, double *e, double *tau,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, if UPLO = 'U', the diagonal and first superdiagonal of A are overwritten by the corresponding elements of the tridiagonal matrix T, and the elements above the first superdiagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors; if UPLO = 'L', the diagonal and first subdiagonal of A are over- written by the corresponding elements of the tridiagonal matrix T, and the elements below the first subdiagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

D Output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The diagonal elements of the tridiagonal matrix T: D(i) = A(i,i).

E Output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N-1). The off-diagonal elements of the tridiagonal matrix T: E(i) = A(i,i+1) if UPLO = 'U', E(i) = A(i+1,i) if UPLO = 'L'.

TAU Output parameter.

TAU is DOUBLE PRECISION

TAU is an array, dimension (N-1). The scalar factors of the elementary reflectors (see Further Details).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *ssytd2*.

4.17.392 dsytf2

dsytf2 computes the factorization of a real symmetric matrix A using the Bunch-Kaufman diagonal pivoting method:

```
A = U*D*U**T  or  A = L*D*L**T
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, U^T is the transpose of U, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the unblocked version of the algorithm, calling Level 2 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsytf2(UPLO, N, A, LDA, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void dsytf2_(const char *uplo, const armpl_int_t *n, double *a,
             const armpl_int_t *lda, armpl_int_t *ipiv, armpl_int_t *info,
             ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L (see below for further details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D.

If UPLO = 'U': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block.

If UPLO = 'L': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value > 0: if INFO = k, D(k,k) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *csytf2*, *ssytf2* and *zsytf2*.

4.17.393 dsytf2_rook

DSYTF2_ROOK computes the factorization of a real symmetric matrix A using the bounded Bunch-Kaufman ("rook") diagonal pivoting method:

```
A = U*D*U**T  or  A = L*D*L**T
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, U^T is the transpose of U, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the unblocked version of the algorithm, calling Level 2 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine dsytf2_rook(UPLO, N, A, LDA, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void dsytf2_rook_(const char *uplo, const armpl_int_t *n, double *a,
                  const armpl_int_t *lda, armpl_int_t *ipiv,
                  armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L (see below for further details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D.

If UPLO = 'U': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k-1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k-1 and -IPIV(k-1) were inerchaged, D(k-1:k,k-1:k) is a 2-by-2 diagonal block.

If UPLO = 'L': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k+1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k+1 and -IPIV(k+1) were inerchaged, D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value > 0: if INFO = k, D(k,k) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *csytf2_rook*, *ssytf2_rook* and *zsytf2_rook*.

4.17.394 dtfsm

Level 3 BLAS like routine for A in RFP Format.

dtfsm solves the matrix equation

```
op( A )*X = alpha*B  or  X*op( A ) = alpha*B
```

where alpha is a scalar, X and B are m by n matrices, A is a unit, or non-unit, upper or lower triangular matrix and op(A) is one of

```
op( A ) = A   or   op( A ) = A**T.
```

A is in Rectangular Full Packed (RFP) Format.

The matrix X is overwritten on B.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtfsm(TRANSR, SIDE, UPLO, TRANS, DIAG, M, N, ALPHA, A, B, LDB)
```

C specification:

```
#include "armpl.h"

void dtfsm_(const char *transr, const char *side, const char *uplo,
            const char *trans, const char *diag, const armpl_int_t *m,
            const armpl_int_t *n, const double *alpha, const double *a,
            double *b, const armpl_int_t *ldb, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': The Normal Form of RFP A is stored; = 'T': The Transpose Form of RFP A is stored.

SIDE Input parameter.

SIDE is CHARACTER*1

On entry, SIDE specifies whether op(A) appears on the left or right of X as follows:

SIDE = 'L' or 'l' op(A)*X = alpha*B.

SIDE = 'R' or 'r' X*op(A) = alpha*B.

Unchanged on exit.

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the RFP matrix A came from an upper or lower triangular matrix as follows: UPLO = 'U' or 'u' RFP A came from an upper triangular matrix UPLO = 'L' or 'l' RFP A came from a lower triangular matrix

Unchanged on exit.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the form of op(A) to be used in the matrix multiplication as follows:

TRANS = 'N' or 'n' op(A) = A.

TRANS = 'T' or 't' op(A) = A'.

Unchanged on exit.

DIAG Input parameter.

DIAG is CHARACTER*1

On entry, DIAG specifies whether or not RFP A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

Unchanged on exit.

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of B. M must be at least zero. Unchanged on exit.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of B. N must be at least zero. Unchanged on exit.

ALPHA Input parameter.

ALPHA is DOUBLE PRECISION

On entry, ALPHA specifies the scalar alpha. When alpha is zero then A is not referenced and B need not be set before entry. Unchanged on exit.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (NT). NT = N*(N+1)/2. On entry, the matrix A in RFP Format. RFP Format is described by TRANSR, UPLO and N as follows: If TRANSR='N' then RFP A is (0:N,0:K-1) when N is even; K=N/2. RFP A is (0:N-1,0:K) when N is odd; K=N/2. If TRANSR = 'T' then RFP is the transpose of RFP A as defined when TRANSR = 'N'. The contents of RFP A are defined by UPLO as follows: If UPLO = 'U' the RFP A contains the NT elements of upper packed A either in normal or transpose Format. If UPLO = 'L' the RFP A contains the NT elements of lower packed A either in normal or transpose Format. The LDA of RFP

A is (N+1)/2 when TRANSR = 'T'. When TRANSR is 'N' the LDA is N+1 when N is even and is N when is odd. See the Note below for more details. Unchanged on exit.

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). Before entry, the leading m by n part of the array B must contain the right-hand side matrix B, and on exit is overwritten by the solution matrix X.

LDB Input parameter.

LDB is INTEGER

On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. LDB must be at least max(1, m). Unchanged on exit.

Related Information

For this routine in other precisions, please see *ctfsm*, *stfsm* and *ztfsm*. It also exists with a native C interface as *LAPACKE_dtfsm*.

4.17.395 dtfttp

dtfttp copies a triangular matrix A from rectangular full packed format (TF) to standard packed format (TP).

Syntax

Fortran specification:

```
use armpl_library

subroutine dtfttp(TRANSR, UPLO, N, ARF, AP, INFO)
```

C specification:

```
#include "armpl.h"

void dtfttp_(const char *transr, const char *uplo, const armpl_int_t *n,
             const double *arf, double *ap, armpl_int_t *info, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': ARF is in Normal format; = 'T': ARF is in Transpose format;

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

ARF Input parameter.

ARF is DOUBLE PRECISION

ARF is an array, dimension (N*(N+1)/2),. On entry, the upper or lower triangular matrix A stored in RFP format. For a further discussion see Notes below.

AP Output parameter.

AP is DOUBLE PRECISION

AP is an array, dimension (N*(N+1)/2),. On exit, the upper or lower triangular matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctfttp*, *stfttp* and *ztfttp*. It also exists with a native C interface as *LAPACKE_dtfttp*.

4.17.396 dtfttr

dtfttr copies a triangular matrix A from rectangular full packed format (TF) to standard full format (TR).

Syntax

Fortran specification:

```
use armpl_library

subroutine dtfttr(TRANSR, UPLO, N, ARF, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void dtfttr_(const char *transr, const char *uplo, const armpl_int_t *n,
             const double *arf, double *a, const armpl_int_t *lda,
             armpl_int_t *info, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': ARF is in Normal format; = 'T': ARF is in Transpose format.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

N Input parameter.

N is INTEGER

The order of the matrices ARF and A. N >= 0.

ARF Input parameter.

ARF is DOUBLE PRECISION

ARF is an array, dimension (N*(N+1)/2).. On entry, the upper (if UPLO = 'U') or lower (if UPLO = 'L') matrix A in RFP format. See the "Notes" below for more details.

A Output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On exit, the triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctfttr*, *stfttr* and *ztfttr*. It also exists with a native C interface as *LAPACKE_dtfttr*.

4.17.397 dtgex2

dtgex2 swaps adjacent diagonal blocks (A11, B11) and (A22, B22) of size 1-by-1 or 2-by-2 in an upper (quasi) triangular matrix pair (A, B) by an orthogonal equivalence transformation.

(A, B) must be in generalized real Schur canonical form (as returned by DGGES), i.e. A is block upper triangular with 1-by-1 and 2-by-2 diagonal blocks. B is upper triangular.

Optionally, the matrices Q and Z of generalized Schur vectors are updated.

```
Q(in) * A(in) * Z(in)**T = Q(out) * A(out) * Z(out)**T
Q(in) * B(in) * Z(in)**T = Q(out) * B(out) * Z(out)**T
```

Syntax

Fortran specification:

```
use armpl_library

subroutine dtgex2(WANTQ, WANTZ, N, A, LDA, B, LDB, Q, LDQ, Z, LDZ, J1, N1, N2,
                  WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void dtgex2_(const armpl_int_t *wantq, const armpl_int_t *wantz,
             const armpl_int_t *n, double *a, const armpl_int_t *lda,
             double *b, const armpl_int_t *ldb, double *q,
             const armpl_int_t *ldq, double *z, const armpl_int_t *ldz,
             const armpl_int_t *j1, const armpl_int_t *n1,
             const armpl_int_t *n2, double *work, const armpl_int_t *lwork,
             armpl_int_t *info);
```

Parameters

WANTQ Input parameter.

WANTQ is LOGICAL

.TRUE. : update the left transformation matrix Q; .FALSE.: do not update Q.

WANTZ Input parameter.

WANTZ is LOGICAL

.TRUE. : update the right transformation matrix Z; .FALSE.: do not update Z.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimensions (LDA, N). On entry, the matrix A in the pair (A, B). On exit, the updated matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimensions (LDB, N). On entry, the matrix B in the pair (A, B). On exit, the updated matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Q Input and output parameter.

Q is DOUBLE PRECISION

Q is an array, dimension (LDQ, N). On entry, if WANTQ = .TRUE., the orthogonal matrix Q. On exit, the updated matrix Q. Not referenced if WANTQ = .FALSE..

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= 1. If WANTQ = .TRUE., LDQ >= N.

Z Input and output parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDZ, N). On entry, if WANTZ =.TRUE., the orthogonal matrix Z. On exit, the updated matrix Z. Not referenced if WANTZ = .FALSE..

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1. If WANTZ = .TRUE., LDZ >= N.

J1 Input parameter.

J1 is INTEGER

The index to the first block (A11, B11). 1 <= J1 <= N.

N1 Input parameter.

N1 is INTEGER

The order of the first block (A11, B11). N1 = 0, 1 or 2.

N2 Input parameter.

N2 is INTEGER

The order of the second block (A22, B22). N2 = 0, 1 or 2.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1, LWORK)). .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= MAX(1, N*(N2+N1), (N2+N1)*(N2+N1)*2)

INFO Output parameter.

INFO is INTEGER

=0: Successful exit >0: If INFO = 1, the transformed matrix (A, B) would be too far from generalized Schur form; the blocks are not swapped and (A, B) and (Q, Z) are unchanged. The problem of swapping is too ill-conditioned. <0: If INFO = -16: LWORK is too small. Appropriate value for LWORK is returned in WORK(1).

Related Information

For this routine in other precisions, please see *ctgex2*, *stgex2* and *ztgex2*.

4.17.398 dtgsy2

dtgsy2 solves the generalized Sylvester equation:

```
A * R - L * B = scale * C                (1)
D * R - L * E = scale * F,
```

using Level 1 and 2 BLAS. where R and L are unknown M-by-N matrices, (A, D), (B, E) and (C, F) are given matrix pairs of size M-by-M, N-by-N and M-by-N, respectively, with real entries. (A, D) and (B, E) must be in generalized Schur canonical form, i.e. A, B are upper quasi triangular and D, E are upper triangular. The solution (R, L) overwrites (C, F). 0 <= SCALE <= 1 is an output scaling factor chosen to avoid overflow.

In matrix notation solving equation (1) corresponds to solve Z*x = scale*b, where Z is defined as

```
Z = [ kron(In, A)  -kron(B**T, Im) ]             (2)
    [ kron(In, D)  -kron(E**T, Im) ],
```

Ik is the identity matrix of size k and X^T is the transpose of X. kron(X, Y) is the Kronecker product between the matrices X and Y. In the process of solving (1), we solve a number of such systems where Dim(In), Dim(In) = 1 or 2.

If TRANS = 'T', solve the transposed system Z^T *y = scale*b for y, which is equivalent to solve for R and L in

```
A**T * R  + D**T * L   = scale * C           (3)
R  * B**T + L  * E**T  = scale * -F
```

This case is used to compute an estimate of Dif[(A, D), (B, E)] = sigma_min(Z) using reverse communicaton with DLACON.

dtgsy2 also (IJOB >= 1) contributes to the computation in DTGSYL of an upper bound on the separation between to matrix pairs. Then the input (A, D), (B, E) are sub-pencils of the matrix pair in DTGSYL. See DTGSYL for details.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtgsy2(TRANS, IJOB, M, N, A, LDA, B, LDB, C, LDC, D, LDD, E, LDE,
                  F, LDF, SCALE, RDSUM, RDSCAL, IWORK, PQ, INFO)
```

C specification:

```
#include "armpl.h"

void dtgsy2_(const char *trans, const armpl_int_t *ijob, const armpl_int_t *m,
             const armpl_int_t *n, const double *a, const armpl_int_t *lda,
             const double *b, const armpl_int_t *ldb, double *c,
             const armpl_int_t *ldc, const double *d, const armpl_int_t *ldd,
             const double *e, const armpl_int_t *lde, double *f,
             const armpl_int_t *ldf, double *scale, double *rdsum,
             double *rdscal, armpl_int_t *iwork, armpl_int_t *pq,
             armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N', solve the generalized Sylvester equation (1). = 'T': solve the 'transposed' system (3).

IJOB Input parameter.

IJOB is INTEGER

Specifies what kind of functionality to be performed. = 0: solve (1) only. = 1: A contribution from this subsystem to a Frobenius norm-based estimate of the separation between two matrix pairs is computed. (look ahead strategy is used). = 2: A contribution from this subsystem to a Frobenius norm-based estimate of the separation between two matrix pairs is computed. (DGECON on sub-systems is used.) Not referenced if TRANS = 'T'.

M Input parameter.

M is INTEGER

On entry, M specifies the order of A and D, and the row dimension of C, F, R and L.

N Input parameter.

N is INTEGER

On entry, N specifies the order of B and E, and the column dimension of C, F, R and L.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, M). On entry, A contains an upper quasi triangular matrix.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the matrix A. LDA >= max(1, M).

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). On entry, B contains an upper quasi triangular matrix.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the matrix B. LDB >= max(1, N).

C Input and output parameter.

C is DOUBLE PRECISION

C is an array, dimension (LDC, N). On entry, C contains the right-hand-side of the first matrix equation in (1). On exit, if IJOB = 0, C has been overwritten by the solution R.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the matrix C. LDC >= max(1, M).

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (LDD, M). On entry, D contains an upper triangular matrix.

LDD Input parameter.

LDD is INTEGER

The leading dimension of the matrix D. LDD >= max(1, M).

E Input parameter.

E is DOUBLE PRECISION

E is an array, dimension (LDE, N). On entry, E contains an upper triangular matrix.

LDE Input parameter.

LDE is INTEGER

The leading dimension of the matrix E. LDE >= max(1, N).

F Input and output parameter.

F is DOUBLE PRECISION

F is an array, dimension (LDF, N). On entry, F contains the right-hand-side of the second matrix equation in (1). On exit, if IJOB = 0, F has been overwritten by the solution L.

LDF Input parameter.

LDF is INTEGER

The leading dimension of the matrix F. LDF >= max(1, M).

SCALE Output parameter.

SCALE is DOUBLE PRECISION

On exit, 0 <= SCALE <= 1. If 0 < SCALE < 1, the solutions R and L (C and F on entry) will hold the solutions to a slightly perturbed system but the input matrices A, B, D and E have not been changed. If SCALE = 0, R and L will hold the solutions to the homogeneous system with C = F = 0. Normally, SCALE = 1.

RDSUM Input and output parameter.

RDSUM is DOUBLE PRECISION

On entry, the sum of squares of computed contributions to the Dif-estimate under computation by DTGSYL, where the scaling factor RDSCAL (see below) has been factored out. On exit, the corresponding sum of squares updated with the contributions from the current sub-system. If TRANS = 'T' RDSUM is not touched. NOTE: RDSUM only makes sense when DTGSY2 is called by DTGSYL.

RDSCAL Input and output parameter.

RDSCAL is DOUBLE PRECISION

On entry, scaling factor used to prevent overflow in RDSUM. On exit, RDSCAL is updated w.r.t. the current contributions in RDSUM. If TRANS = 'T', RDSCAL is not touched. NOTE: RDSCAL only makes sense when DTGSY2 is called by DTGSYL.

IWORK Output parameter.

IWORK is INTEGER array, dimension (M+N+2)

PQ Output parameter.

PQ is INTEGER

On exit, the number of subsystems (of size 2-by-2, 4-by-4 and 8-by-8) solved by this routine.

INFO Output parameter.

INFO is INTEGER

On exit, if INFO is set to =0: Successful exit <0: If INFO = -i, the i-th argument had an illegal value. >0: The matrix pairs (A, D) and (B, E) have common or very close eigenvalues.

Related Information

For this routine in other precisions, please see *ctgsy2*, *stgsy2* and *ztgsy2*.

4.17.399 dtplqt2

dtplqt2 computes a LQ a factorization of a real "triangular-pentagonal" matrix C, which is composed of a triangular block A and pentagonal block B, using the compact WY representation for Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtplqt2(M, N, L, A, LDA, B, LDB, T, LDT, INFO)
```

C specification:

```
#include "armpl.h"

void dtplqt2_(const armpl_int_t *m, const armpl_int_t *n,
              const armpl_int_t *l, double *a, const armpl_int_t *lda,
              double *b, const armpl_int_t *ldb, double *t,
              const armpl_int_t *ldt, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The total number of rows of the matrix B. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix B, and the order of the triangular matrix A. N >= 0.

L Input parameter.

L is INTEGER

The number of rows of the lower trapezoidal part of B. MIN(M, N) >= L >= 0. See Further Details.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, M). On entry, the lower triangular M-by-M matrix A. On exit, the elements on and below the diagonal of the array contain the lower triangular matrix L.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). On entry, the pentagonal M-by-N matrix B. The first N-L columns are rectangular, and the last L columns are lower trapezoidal. On exit, B contains the pentagonal matrix V. See Further Details.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M).

T Output parameter.

T is DOUBLE PRECISION

T is an array, dimension (LDT, M). The N-by-N upper triangular factor T of the block reflector. See Further Details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, M)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctplqt2*, *stplqt2* and *ztplqt2*.

4.17.400 dtpqrt2

dtpqrt2 computes a QR factorization of a real "triangular-pentagonal" matrix C, which is composed of a triangular block A and pentagonal block B, using the compact WY representation for Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtpqrt2(M, N, L, A, LDA, B, LDB, T, LDT, INFO)
```

C specification:

```
#include "armpl.h"

void dtpqrt2_(const armpl_int_t *m, const armpl_int_t *n,
              const armpl_int_t *l, double *a, const armpl_int_t *lda,
              double *b, const armpl_int_t *ldb, double *t,
              const armpl_int_t *ldt, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The total number of rows of the matrix B. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix B, and the order of the triangular matrix A. N >= 0.

L Input parameter.

L is INTEGER

The number of rows of the upper trapezoidal part of B. MIN(M, N) >= L >= 0. See Further Details.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the upper triangular N-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the upper triangular matrix R.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). On entry, the pentagonal M-by-N matrix B. The first M-L rows are rectangular, and the last L rows are upper trapezoidal. On exit, B contains the pentagonal matrix V. See Further Details.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M).

T Output parameter.

T is DOUBLE PRECISION

T is an array, dimension (LDT, N). The N-by-N upper triangular factor T of the block reflector. See Further Details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctpqrt2*, *stpqrt2* and *ztpqrt2*. It also exists with a native C interface as *LAPACKE_dtpqrt2*.

4.17.401 dtprfb

dtprfb applies a real "triangular-pentagonal" block reflector H or its transpose H^T to a real matrix C, which is composed of two blocks A and B, either from the left or right.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtprfb(SIDE, TRANS, DIRECT, STOREV, M, N, K, L, V, LDV, T, LDT, A,
                  LDA, B, LDB, WORK, LDWORK)
```

C specification:

```
#include "armpl.h"

void dtprfb_(const char *side, const char *trans, const char *direct,
             const char *storev, const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *k, const armpl_int_t *l, const double *v,
             const armpl_int_t *ldv, const double *t, const armpl_int_t *ldt,
             double *a, const armpl_int_t *lda, double *b,
             const armpl_int_t *ldb, double *work, const armpl_int_t *ldwork,
             ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply H or H^T from the Left = 'R': apply H or H^T from the Right

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply H (No transpose) = 'T': apply H^T (Transpose)

DIRECT Input parameter.

DIRECT is CHARACTER*1

Indicates how H is formed from a product of elementary reflectors = 'F': H = H(1) H(2) . . . H(k) (Forward) = 'B': H = H(k) . . . H(2) H(1) (Backward)

STOREV Input parameter.

STOREV is CHARACTER*1

Indicates how the vectors which define the elementary reflectors are stored: = 'C': Columns = 'R': Rows

M Input parameter.

M is INTEGER

The number of rows of the matrix B. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix B. N >= 0.

K Input parameter.

K is INTEGER

The order of the matrix T, i.e. the number of elementary reflectors whose product defines the block reflector. K >= 0.

L Input parameter.

L is INTEGER

The order of the trapezoidal part of V. K >= L >= 0. See Further Details.

V Input parameter.

V is DOUBLE PRECISION

V is an array, dimension. (LDV, K) if STOREV = 'C' (LDV, M) if STOREV = 'R' and SIDE = 'L' (LDV, N) if STOREV = 'R' and SIDE = 'R' The pentagonal matrix V, which contains the elementary reflectors H(1), H(2), . . . , H(K). See Further Details.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. If STOREV = 'C' and SIDE = 'L', LDV >= max(1, M); if STOREV = 'C' and SIDE = 'R', LDV >= max(1, N); if STOREV = 'R', LDV >= K.

T Input parameter.

T is DOUBLE PRECISION

T is an array, dimension (LDT, K). The triangular K-by-K matrix T in the representation of the block reflector.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= K.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension. (LDA, N) if SIDE = 'L' or (LDA, K) if SIDE = 'R' On entry, the K-by-N or M-by-K matrix A. On exit, A is overwritten by the corresponding block of H*C or H^T *C or C*H or C*H^T . See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDC >= max(1, K); If SIDE = 'R', LDC >= max(1, M).

B Input and output parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). On entry, the M-by-N matrix B. On exit, B is overwritten by the corresponding block of H*C or H^T *C or C*H or C*H^T . See Further Details.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension. (LDWORK, N) if SIDE = 'L', (LDWORK, K) if SIDE = 'R'.

LDWORK Input parameter.

LDWORK is INTEGER

The leading dimension of the array WORK. If SIDE = 'L', LDWORK >= K; if SIDE = 'R', LDWORK >= M.

Related Information

For this routine in other precisions, please see *ctprfb*, *stprfb* and *ztprfb*. It also exists with a native C interface as *LAPACKE_dtprfb*.

4.17.402 dtpttf

dtpttf copies a triangular matrix A from standard packed format (TP) to rectangular full packed format (TF).

Syntax

Fortran specification:

```
use armpl_library

subroutine dtpttf(TRANSR, UPLO, N, AP, ARF, INFO)
```

C specification:

```
#include "armpl.h"

void dtpttf_(const char *transr, const char *uplo, const armpl_int_t *n,
             const double *ap, double *arf, armpl_int_t *info, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': ARF in Normal format is wanted; = 'T': ARF in Conjugate-transpose format is wanted.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input parameter.

AP is DOUBLE PRECISION

AP is an array, dimension (N*(N+1)/2),. On entry, the upper or lower triangular matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

ARF Output parameter.

ARF is DOUBLE PRECISION

ARF is an array, dimension (N*(N+1)/2),. On exit, the upper or lower triangular matrix A stored in RFP format. For a further discussion see Notes below.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctpttf*, *stpttf* and *ztpttf*. It also exists with a native C interface as *LAPACKE_dtpttf*.

4.17.403 dtpttr

dtpttr copies a triangular matrix A from standard packed format (TP) to standard full format (TR).

Syntax

Fortran specification:

```
use armpl_library

subroutine dtpttr(UPLO, N, AP, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void dtpttr_(const char *uplo, const armpl_int_t *n, const double *ap,
             double *a, const armpl_int_t *lda, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular. = 'L': A is lower triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input parameter.

AP is DOUBLE PRECISION

AP is an array, dimension (N*(N+1)/2),. On entry, the upper or lower triangular matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

A Output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On exit, the triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctpttr*, *stpttr* and *ztpttr*. It also exists with a native C interface as *LAPACKE_dtpttr*.

4.17.404 dtrti2

dtrti2 computes the inverse of a real upper or lower triangular matrix.

This is the Level 2 BLAS version of the algorithm.

Syntax

Fortran specification:

```
use armpl_library

subroutine dtrti2(UPLO, DIAG, N, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void dtrti2_(const char *uplo, const char *diag, const armpl_int_t *n,
             double *a, const armpl_int_t *lda, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the matrix A is upper or lower triangular. = 'U': Upper triangular = 'L': Lower triangular

DIAG Input parameter.

DIAG is CHARACTER*1

Specifies whether or not the matrix A is unit triangular. = 'N': Non-unit triangular = 'U': Unit triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On entry, the triangular matrix A. If UPLO = 'U', the leading n by n upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n by n lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1.

On exit, the (triangular) inverse of the original matrix, in the same storage format.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctrti2*, *strti2* and *ztrti2*.

4.17.405 dtrttf

dtrttf copies a triangular matrix A from standard full format (TR) to rectangular full packed format (TF) .

Syntax

Fortran specification:

```
use armpl_library

subroutine dtrttf(TRANSR, UPLO, N, A, LDA, ARF, INFO)
```

C specification:

```
#include "armpl.h"

void dtrttf_(const char *transr, const char *uplo, const armpl_int_t *n,
             const double *a, const armpl_int_t *lda, double *arf,
             armpl_int_t *info, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': ARF in Normal form is wanted; = 'T': ARF in Transpose form is wanted.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N).. On entry, the triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the matrix A. LDA >= max(1, N).

ARF Output parameter.

ARF is DOUBLE PRECISION

ARF is an array, dimension (NT).. NT=N*(N+1)/2. On exit, the triangular matrix A in RFP format.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctrttf*, *strttf* and *ztrttf*. It also exists with a native C interface as *LAPACKE_dtrttf*.

4.17.406 dtrttp

dtrttp copies a triangular matrix A from full format (TR) to standard packed format (TP).

Syntax

Fortran specification:

```
use armpl_library

subroutine dtrttp(UPLO, N, A, LDA, AP, INFO)
```

C specification:

```
#include "armpl.h"

void dtrttp_(const char *uplo, const armpl_int_t *n, const double *a,
             const armpl_int_t *lda, double *ap, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular. = 'L': A is lower triangular.

N Input parameter.

N is INTEGER

The order of the matrices AP and A. N >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On exit, the triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AP Output parameter.

AP is DOUBLE PRECISION

AP is an array, dimension (N*(N+1)/2). On exit, the upper or lower triangular matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctrttp*, *strttp* and *ztrttp*. It also exists with a native C interface as *LAPACKE_dtrttp*.

4.17.407 dzsum1

`dzsum1` takes the sum of the absolute values of a complex vector and returns a double precision result.

Based on DZASUM from the Level 1 BLAS. The change is to use the 'genuine' absolute value.

Syntax

Fortran specification:

```
use armpl_library

double precision function dzsum1(N, CX, INCX)
```

C specification:

```
#include "armpl.h"

double dzsum1_(const armpl_int_t *n, const armpl_doublecomplex_t *cx,
               const armpl_int_t *incx);
```

Parameters

N Input parameter.

`N` is `INTEGER`

The number of elements in the vector `CX`.

CX Input parameter.

`CX` is `COMPLEX*16`

`CX` is an array, dimension (`N`). The vector whose elements will be summed.

INCX Input parameter.

`INCX` is `INTEGER`

The spacing between successive values of `CX`. `INCX` > 0.

Related Information

4.17.408 scsum1

`scsum1` takes the sum of the absolute values of a complex vector and returns a single precision result.

Based on SCASUM from the Level 1 BLAS. The change is to use the 'genuine' absolute value.

Syntax

Fortran specification:

```
use armpl_library

real function scsum1(N, CX, INCX)
```

C specification:

```
#include "armpl.h"

float scsum1_(const armpl_int_t *n, const armpl_singlecomplex_t *cx,
              const armpl_int_t *incx);
```

Parameters

N Input parameter.

N is INTEGER

The number of elements in the vector CX.

CX Input parameter.

CX is COMPLEX

CX is an array, dimension (N). The vector whose elements will be summed.

INCX Input parameter.

INCX is INTEGER

The spacing between successive values of CX. INCX > 0.

Related Information

4.17.409 sgbtf2

sgbtf2 computes an LU factorization of a real m-by-n band matrix A using partial pivoting with row interchanges. This is the unblocked version of the algorithm, calling Level 2 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgbtf2(M, N, KL, KU, AB, LDAB, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void sgbtf2_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *kl, const armpl_int_t *ku, float *ab,
             const armpl_int_t *ldab, armpl_int_t *ipiv, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

AB Input and output parameter.

AB is REAL

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows KL+1 to 2*KL+KU+1; rows 1 to KL of the array need not be set. The j-th column of A is stored in the j-th column of the array AB as follows: AB(kl+ku+1+i-j,j) = A(i,j) for max(1,j-ku)<=i<=min(m,j+kl)

On exit, details of the factorization: U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1. See below for further details.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= 2*KL+KU+1.

IPIV Output parameter.

IPIV is INTEGER array, dimension (min(M, N))

The pivot indices; for 1 <= i <= min(M, N), row i of the matrix was interchanged with row IPIV(i).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = +i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *cgbtf2*, *dgbtf2* and *zgbtf2*.

4.17.410 sgebd2

sgebd2 reduces a real general m by n matrix A to upper or lower bidiagonal form B by an orthogonal transformation: $Q^T * A * P = B$.

If m >= n, B is upper bidiagonal; if m < n, B is lower bidiagonal.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgebd2(M, N, A, LDA, D, E, TAUQ, TAUP, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgebd2_(const armpl_int_t *m, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, float *d, float *e, float *tauq,
             float *taup, float *work, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows in the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns in the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the m by n general matrix to be reduced. On exit, if m >= n, the diagonal and the first superdiagonal are overwritten with the upper bidiagonal matrix B; the elements below the diagonal, with the array TAUQ, represent the orthogonal matrix Q as a product of elementary reflectors, and the elements above the first superdiagonal, with the array TAUP, represent the orthogonal matrix P as a product of elementary reflectors; if m < n, the diagonal and the first subdiagonal are overwritten with the lower bidiagonal matrix B; the elements below the first subdiagonal, with the array TAUQ, represent the orthogonal matrix Q as a product of elementary reflectors, and the elements above the diagonal, with the array TAUP, represent the orthogonal matrix P as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

D Output parameter.

D is REAL

D is an array, dimension (min(M, N)). The diagonal elements of the bidiagonal matrix B: D(i) = A(i,i).

E Output parameter.

E is REAL

E is an array, dimension (min(M, N)-1). The off-diagonal elements of the bidiagonal matrix B: if m >= n, E(i) = A(i,i+1) for i = 1,2,...,n-1; if m < n, E(i) = A(i+1,i) for i = 1,2,...,m-1.

TAUQ Output parameter.

TAUQ is REAL

TAUQ is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors which represent the orthogonal matrix Q. See Further Details.

TAUP Output parameter.

TAUP is REAL

TAUP is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors which represent the orthogonal matrix P. See Further Details.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (max(M, N)) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cgebd2*, *dgebd2* and *zgebd2*.

4.17.411 sgehd2

sgehd2 reduces a real general matrix A to upper Hessenberg form H by an orthogonal similarity transformation: $Q^T * A * Q = H$.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgehd2(N, ILO, IHI, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgehd2_(const armpl_int_t *n, const armpl_int_t *ilo,
             const armpl_int_t *ihi, float *a, const armpl_int_t *lda,
             float *tau, float *work, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

It is assumed that A is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N. ILO and IHI are normally set by a previous call to SGEBAL; otherwise they should be set to 1 and N respectively. See Further Details. 1 <= ILO <= IHI <= max(1, N).

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the n by n general matrix to be reduced. On exit, the upper triangle and the first subdiagonal of A are overwritten with the upper Hessenberg matrix H, and the elements below the first subdiagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TAU Output parameter.

TAU is REAL

TAU is an array, dimension (N-1). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cgehd2*, *dgehd2* and *zgehd2*.

4.17.412 sgelq2

sgelq2 computes an LQ factorization of a real m by n matrix A: A = L * Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgelq2(M, N, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgelq2_(const armpl_int_t *m, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, float *tau, float *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the m by n matrix A. On exit, the elements on and below the diagonal of the array contain the m by min(m,n) lower trapezoidal matrix L (L is lower triangular if m <= n); the elements above the diagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is REAL

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (M) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgelq2*, *dgelq2* and *zgelq2*. It also exists with a native C interface as *LAPACKE_sgelq2*.

4.17.413 sgelqt3

DGELQT3 recursively computes a LQ factorization of a real M-by-N matrix A, using the compact WY representation of Q.
Based on the algorithm of Elmroth and Gustavson, IBM J. Res. Develop. Vol 44 No. 4 July 2000.

Syntax

Fortran specification:

```
use armpl_library

recursive subroutine sgelqt3(M, N, A, LDA, T, LDT, INFO)
```

C specification:

```
#include "armpl.h"

void sgelqt3_(const armpl_int_t *m, const armpl_int_t *n, float *a,
              const armpl_int_t *lda, float *t, const armpl_int_t *ldt,
              armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M =< N.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the real M-by-N matrix A. On exit, the elements on and below the diagonal contain the N-by-N lower triangular matrix L; the elements above the diagonal are the rows of V. See below for further details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

T Output parameter.

T is REAL

T is an array, dimension (LDT, N). The N-by-N upper triangular factor of the block reflector. The elements on and above the diagonal contain the block reflector T; the elements below the diagonal are not used. See below for further details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgelqt3*, *dgelqt3* and *zgelqt3*.

4.17.414 sgeql2

sgeql2 computes a QL factorization of a real m by n matrix A: A = Q * L.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgeql2(M, N, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgeql2_(const armpl_int_t *m, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, float *tau, float *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the m by n matrix A. On exit, if m >= n, the lower triangle of the subarray A(m-n+1:m,1:n) contains the n by n lower triangular matrix L; if m <= n, the elements on and below the (n-m)-th superdiagonal contain the m by n lower trapezoidal matrix L; the remaining elements, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is REAL

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgeql2*, *dgeql2* and *zgeql2*.

4.17.415 sgeqr2

sgeqr2 computes a QR factorization of a real m by n matrix A: A = Q * R.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgeqr2(M, N, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgeqr2_(const armpl_int_t *m, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, float *tau, float *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the m by n matrix A. On exit, the elements on and above the diagonal of the array contain the min(m,n) by n upper trapezoidal matrix R (R is upper triangular if m >= n); the elements below the diagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is REAL

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgeqr2*, *dgeqr2* and *zgeqr2*. It also exists with a native C interface as *LAPACKE_sgeqr2*.

4.17.416 sgeqr2p

sgeqr2p computes a QR factorization of a real m by n matrix A: A = Q * R. The diagonal entries of R are nonnegative.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgeqr2p(M, N, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgeqr2p_(const armpl_int_t *m, const armpl_int_t *n, float *a,
              const armpl_int_t *lda, float *tau, float *work,
              armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the m by n matrix A. On exit, the elements on and above the diagonal of the array contain the min(m,n) by n upper trapezoidal matrix R (R is upper triangular if m >= n). The diagonal entries of R are nonnegative; the elements below the diagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is REAL

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgeqr2p*, *dgeqr2p* and *zgeqr2p*.

4.17.417 sgeqrt2

`sgeqrt2` computes a QR factorization of a real M-by-N matrix A, using the compact WY representation of Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgeqrt2(M, N, A, LDA, T, LDT, INFO)
```

C specification:

```
#include "armpl.h"

void sgeqrt2_(const armpl_int_t *m, const armpl_int_t *n, float *a,
              const armpl_int_t *lda, float *t, const armpl_int_t *ldt,
              armpl_int_t *info);
```

Parameters

M Input parameter.

`M` is `INTEGER`

The number of rows of the matrix `A`. `M` >= `N`.

N Input parameter.

`N` is `INTEGER`

The number of columns of the matrix `A`. `N` >= 0.

A Input and output parameter.

`A` is `REAL`

`A` is an array, dimension (`LDA`, `N`). On entry, the real M-by-N matrix `A`. On exit, the elements on and above the diagonal contain the N-by-N upper triangular matrix R; the elements below the diagonal are the columns of V. See below for further details.

LDA Input parameter.

`LDA` is `INTEGER`

The leading dimension of the array `A`. `LDA` >= max(1, `M`).

T Output parameter.

`T` is `REAL`

`T` is an array, dimension (`LDT`, `N`). The N-by-N upper triangular factor of the block reflector. The elements on and above the diagonal contain the block reflector T; the elements below the diagonal are not used. See below for further details.

LDT Input parameter.

`LDT` is `INTEGER`

The leading dimension of the array `T`. `LDT` >= max(1, `N`).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgeqrt2*, *dgeqrt2* and *zgeqrt2*. It also exists with a native C interface as *LAPACKE_sgeqrt2*.

4.17.418 sgeqrt3

sgeqrt3 recursively computes a QR factorization of a real M-by-N matrix A, using the compact WY representation of Q.

Based on the algorithm of Elmroth and Gustavson, IBM J. Res. Develop. Vol 44 No. 4 July 2000.

Syntax

Fortran specification:

```
use armpl_library

recursive subroutine sgeqrt3(M, N, A, LDA, T, LDT, INFO)
```

C specification:

```
#include "armpl.h"

void sgeqrt3_(const armpl_int_t *m, const armpl_int_t *n, float *a,
              const armpl_int_t *lda, float *t, const armpl_int_t *ldt,
              armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= N.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the real M-by-N matrix A. On exit, the elements on and above the diagonal contain the N-by-N upper triangular matrix R; the elements below the diagonal are the columns of V. See below for further details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

T Output parameter.

T is REAL

T is an array, dimension (LDT, N). The N-by-N upper triangular factor of the block reflector. The elements on and above the diagonal contain the block reflector T; the elements below the diagonal are not used. See below for further details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgeqrt3*, *dgeqrt3* and *zgeqrt3*. It also exists with a native C interface as *LAPACKE_sgeqrt3*.

4.17.419 sgerq2

sgerq2 computes an RQ factorization of a real m by n matrix A: A = R * Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgerq2(M, N, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgerq2_(const armpl_int_t *m, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, float *tau, float *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the m by n matrix A. On exit, if m <= n, the upper triangle of the subarray A(1:m,n-m+1:n) contains the m by m upper triangular matrix R; if m >= n, the elements on and above the (m-n)-th subdiagonal contain the m by n upper trapezoidal matrix R; the remaining elements, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is REAL

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (M) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgerq2*, *dgerq2* and *zgerq2*.

4.17.420 sgesc2

sgesc2 solves a system of linear equations

```
A * X = scale* RHS
```

with a general N-by-N matrix A using the LU factorization with complete pivoting computed by SGETC2.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgesc2(N, A, LDA, RHS, IPIV, JPIV, SCALE)
```

C specification:

```
#include "armpl.h"

void sgesc2_(const armpl_int_t *n, const float *a, const armpl_int_t *lda,
             float *rhs, const armpl_int_t *ipiv, const armpl_int_t *jpiv,
             float *scale);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the LU part of the factorization of the n-by-n matrix A computed by SGETC2: A = P * L * U * Q

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

RHS Input and output parameter.

RHS is REAL

RHS is an array, dimension (N).. On entry, the right hand side vector b. On exit, the solution vector X.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N).

The pivot indices; for 1 <= i <= N, row i of the matrix has been interchanged with row IPIV(i).

JPIV Input parameter.

JPIV is INTEGER array, dimension (N).

The pivot indices; for 1 <= j <= N, column j of the matrix has been interchanged with column JPIV(j).

SCALE Output parameter.

SCALE is REAL

On exit, SCALE contains the scale factor. SCALE is chosen 0 <= SCALE <= 1 to prevent owerflow in the solution.

Related Information

For this routine in other precisions, please see *cgesc2*, *dgesc2* and *zgesc2*.

4.17.421 sgetc2

sgetc2 computes an LU factorization with complete pivoting of the n-by-n matrix A. The factorization has the form A = P * L * U * Q, where P and Q are permutation matrices, L is lower triangular with unit diagonal elements and U is upper triangular.

This is the Level 2 BLAS algorithm.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgetc2(N, A, LDA, IPIV, JPIV, INFO)
```

C specification:

```
#include "armpl.h"

void sgetc2_(const armpl_int_t *n, float *a, const armpl_int_t *lda,
             armpl_int_t *ipiv, armpl_int_t *jpiv, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the n-by-n matrix A to be factored. On exit, the factors L and U from the factorization A = P*L*U*Q; the unit diagonal elements of L are not stored. If U(k, k) appears to be less than SMIN, U(k, k) is given the value of SMIN, i.e., giving a nonsingular perturbed system.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension(N).

The pivot indices; for 1 <= i <= N, row i of the matrix has been interchanged with row IPIV(i).

JPIV Output parameter.

JPIV is INTEGER array, dimension(N).

The pivot indices; for 1 <= j <= N, column j of the matrix has been interchanged with column JPIV(j).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit > 0: if INFO = k, U(k, k) is likely to produce owerflow if we try to solve for x in Ax = b. So U is perturbed to avoid the overflow.

Related Information

For this routine in other precisions, please see *cgetc2*, *dgetc2* and *zgetc2*.

4.17.422 sgetf2

sgetf2 computes an LU factorization of a general m-by-n matrix A using partial pivoting with row interchanges.

The factorization has the form

```
A = P * L * U
```

where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if m > n), and U is upper triangular (upper trapezoidal if m < n).

This is the right-looking Level 2 BLAS version of the algorithm.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgetf2(M, N, A, LDA, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void sgetf2_(const armpl_int_t *m, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, armpl_int_t *ipiv, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the m by n matrix to be factored. On exit, the factors L and U from the factorization A = P*L*U; the unit diagonal elements of L are not stored.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

IPIV Output parameter.

IPIV is INTEGER array, dimension (min(M, N))

The pivot indices; for 1 <= i <= min(M, N), row i of the matrix was interchanged with row IPIV(i).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value > 0: if INFO = k, U(k,k) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *cgetf2*, *dgetf2* and *zgetf2*. It also exists with a native C interface as *LAPACKE_sgetf2*.

4.17.423 sgsvj0

sgsvj0 is called from SGESVJ as a pre-processor and that is its main purpose. It applies Jacobi rotations in the same way as SGESVJ does, but it does not check convergence (stopping criterion). Few tuning parameters (marked by [TP]) are available for the implementer.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgsvj0(JOBV, M, N, A, LDA, D, SVA, MV, V, LDV, EPS, SFMIN, TOL,
                  NSWEEP, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgsvj0_(const char *jobv, const armpl_int_t *m, const armpl_int_t *n,
             float *a, const armpl_int_t *lda, float *d, float *sva,
             const armpl_int_t *mv, float *v, const armpl_int_t *ldv,
             const float *eps, const float *sfmin, const float *tol,
             const armpl_int_t *nsweep, float *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

JOBV Input parameter.

JOBV is CHARACTER*1

Specifies whether the output from this procedure is used to compute the matrix V: = 'V': the product of the Jacobi rotations is accumulated by postmulyiplying the N-by-N array V. (See the description of V.) = 'A': the product of the Jacobi rotations is accumulated by postmulyiplying the MV-by-N array V. (See the descriptions of MV and V.) = 'N': the Jacobi rotations are not accumulated.

M Input parameter.

M is INTEGER

The number of rows of the input matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the input matrix A. M >= N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, M-by-N matrix A, such that A*diag(D) represents the input matrix. On exit, A_onexit * D_onexit represents the input matrix A*diag(D) post-multiplied by a sequence of Jacobi rotations, where the rotation threshold and the total number of sweeps are given in TOL and NSWEEP, respectively. (See the descriptions of D, TOL and NSWEEP.)

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

D Input and output parameter.

D is REAL

D is an array, dimension (N). The array D accumulates the scaling factors from the fast scaled Jacobi rotations. On entry, A*diag(D) represents the input matrix. On exit, A_onexit*diag(D_onexit) represents the input matrix post-multiplied by a sequence of Jacobi rotations, where the rotation threshold and the total number of sweeps are given in TOL and NSWEEP, respectively. (See the descriptions of A, TOL and NSWEEP.)

SVA Input and output parameter.

SVA is REAL

SVA is an array, dimension (N). On entry, SVA contains the Euclidean norms of the columns of the matrix A*diag(D). On exit, SVA contains the Euclidean norms of the columns of the matrix onexit*diag(D_onexit).

MV Input parameter.

MV is INTEGER

If JOBV .EQ. 'A', then MV rows of V are post-multipled by a sequence of Jacobi rotations. If JOBV = 'N', then MV is not referenced.

V Input and output parameter.

V is REAL

V is an array, dimension (LDV, N). If JOBV .EQ. 'V' then N rows of V are post-multipled by a sequence of Jacobi rotations. If JOBV .EQ. 'A' then MV rows of V are post-multipled by a sequence of Jacobi rotations. If JOBV = 'N', then V is not referenced.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V, LDV >= 1. If JOBV = 'V', LDV .GE. N. If JOBV = 'A', LDV .GE. MV.

EPS Input parameter.

EPS is REAL

EPS = SLAMCH('Epsilon')

SFMIN Input parameter.

SFMIN is REAL

SFMIN = SLAMCH('Safe Minimum')

TOL Input parameter.

TOL is REAL

TOL is the threshold for Jacobi rotations. For a pair A(:,p), A(:,q) of pivot columns, the Jacobi rotation is applied only if ABS(COS(angle(A(:,p),A(:,q)))) .GT. TOL.

NSWEEP Input parameter.

NSWEEP is INTEGER

NSWEEP is the number of sweeps of Jacobi rotations to be performed.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

LWORK is the dimension of WORK. LWORK .GE. M.

INFO Output parameter.

INFO is INTEGER

= 0 : successful exit. < 0 : if INFO = -i, then the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgsvj0*, *dgsvj0* and *zgsvj0*.

4.17.424 sgsvj1

`sgsvj1` is called from SGESVJ as a pre-processor and that is its main purpose. It applies Jacobi rotations in the same way as SGESVJ does, but it targets only particular pivots and it does not check convergence (stopping criterion). Few tunning parameters (marked by [TP]) are available for the implementer.

Further Details

`sgsvj1` applies few sweeps of Jacobi rotations in the column space of the input M-by-N matrix A. The pivot pairs are taken from the (1,2) off-diagonal block in the corresponding N-by-N Gram matrix A^T * A. The block-entries (tiles) of the (1,2) off-diagonal block are marked by the [x]'s in the following scheme:

```
| *  *  * [x] [x] [x]|
| *  *  * [x] [x] [x]|    Row-cycling in the nblr-by-nblc [x] blocks.
| *  *  * [x] [x] [x]|    Row-cyclic pivoting inside each [x] block.
|[x] [x] [x] *  *  * |
|[x] [x] [x] *  *  * |
|[x] [x] [x] *  *  * |
```

In terms of the columns of A, the first N1 columns are rotated 'against' the remaining N-N1 columns, trying to increase the angle between the corresponding subspaces. The off-diagonal block is N1-by(N-N1) and it is tiled using quadratic tiles of side KBL. Here, KBL is a tunning parmeter. The number of sweeps is given in NSWEEP and the orthogonality threshold is given in TOL.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgsvj1(JOBV, M, N, N1, A, LDA, D, SVA, MV, V, LDV, EPS, SFMIN, TOL,
                  NSWEEP, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sgsvj1_(const char *jobv, const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *n1, float *a, const armpl_int_t *lda,
             float *d, float *sva, const armpl_int_t *mv, float *v,
             const armpl_int_t *ldv, const float *eps, const float *sfmin,
             const float *tol, const armpl_int_t *nsweep, float *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

JOBV Input parameter.

JOBV is CHARACTER*1

Specifies whether the output from this procedure is used to compute the matrix V: = 'V': the product of the Jacobi rotations is accumulated by postmulyiplying the N-by-N array V. (See the description of V.) =

'A': the product of the Jacobi rotations is accumulated by postmulyiplying the MV-by-N array V. (See the descriptions of MV and V.) = 'N': the Jacobi rotations are not accumulated.

M Input parameter.

M is INTEGER

The number of rows of the input matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the input matrix A. M >= N >= 0.

N1 Input parameter.

N1 is INTEGER

N1 specifies the 2 x 2 block partition, the first N1 columns are rotated 'against' the remaining N-N1 columns of A.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, M-by-N matrix A, such that A*diag(D) represents the input matrix. On exit, A_onexit * D_onexit represents the input matrix A*diag(D) post-multiplied by a sequence of Jacobi rotations, where the rotation threshold and the total number of sweeps are given in TOL and NSWEEP, respectively. (See the descriptions of N1, D, TOL and NSWEEP.)

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

D Input and output parameter.

D is REAL

D is an array, dimension (N). The array D accumulates the scaling factors from the fast scaled Jacobi rotations. On entry, A*diag(D) represents the input matrix. On exit, A_onexit*diag(D_onexit) represents the input matrix post-multiplied by a sequence of Jacobi rotations, where the rotation threshold and the total number of sweeps are given in TOL and NSWEEP, respectively. (See the descriptions of N1, A, TOL and NSWEEP.)

SVA Input and output parameter.

SVA is REAL

SVA is an array, dimension (N). On entry, SVA contains the Euclidean norms of the columns of the matrix A*diag(D). On exit, SVA contains the Euclidean norms of the columns of the matrix onexit*diag(D_onexit).

MV Input parameter.

MV is INTEGER

If JOBV .EQ. 'A', then MV rows of V are post-multipled by a sequence of Jacobi rotations. If JOBV = 'N', then MV is not referenced.

V Input and output parameter.

V is REAL

V is an array, dimension (LDV, N). If JOBV .EQ. 'V' then N rows of V are post-multipled by a sequence of Jacobi rotations. If JOBV .EQ. 'A' then MV rows of V are post-multipled by a sequence of Jacobi rotations. If JOBV = 'N', then V is not referenced.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V, LDV >= 1. If JOBV = 'V', LDV .GE. N. If JOBV = 'A', LDV .GE. MV.

EPS Input parameter.

EPS is REAL

EPS = SLAMCH('Epsilon')

SFMIN Input parameter.

SFMIN is REAL

SFMIN = SLAMCH('Safe Minimum')

TOL Input parameter.

TOL is REAL

TOL is the threshold for Jacobi rotations. For a pair A(:,p), A(:,q) of pivot columns, the Jacobi rotation is applied only if ABS(COS(angle(A(:,p),A(:,q)))) .GT. TOL.

NSWEEP Input parameter.

NSWEEP is INTEGER

NSWEEP is the number of sweeps of Jacobi rotations to be performed.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

LWORK is the dimension of WORK. LWORK .GE. M.

INFO Output parameter.

INFO is INTEGER

= 0 : successful exit. < 0 : if INFO = -i, then the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgsvj1*, *dgsvj1* and *zgsvj1*.

4.17.425 sgtts2

sgtts2 solves one of the systems of equations

```
A*X = B  or  A**T*X = B,
```

with a tridiagonal matrix A using the LU factorization computed by SGTTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine sgtts2(ITRANS, N, NRHS, DL, D, DU, DU2, IPIV, B, LDB)
```

C specification:

```
#include "armpl.h"

void sgtts2_(const armpl_int_t *itrans, const armpl_int_t *n,
             const armpl_int_t *nrhs, const float *dl, const float *d,
             const float *du, const float *du2, const armpl_int_t *ipiv,
             float *b, const armpl_int_t *ldb);
```

Parameters

ITRANS Input parameter.

ITRANS is INTEGER

Specifies the form of the system of equations. = 0: A * X = B (No transpose) = 1: A^T * X = B (Transpose) = 2: A^T * X = B (Conjugate transpose = Transpose)

N Input parameter.

N is INTEGER

The order of the matrix A.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

DL Input parameter.

DL is REAL

DL is an array, dimension (N-1). The (n-1) multipliers that define the matrix L from the LU factorization of A.

D Input parameter.

D is REAL

D is an array, dimension (N). The n diagonal elements of the upper triangular matrix U from the LU factorization of A.

DU Input parameter.

DU is REAL

DU is an array, dimension (N-1). The (n-1) elements of the first super-diagonal of U.

DU2 Input parameter.

DU2 is REAL

DU2 is an array, dimension (N-2). The (n-2) elements of the second super-diagonal of U.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices; for 1 <= i <= n, row i of the matrix was interchanged with row IPIV(i). IPIV(i) will always be either i or i+1; IPIV(i) = i indicates a row interchange was not required.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the matrix of right hand side vectors B. On exit, B is overwritten by the solution vectors X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Related Information

For this routine in other precisions, please see *cgtts2*, *dgtts2* and *zgtts2*.

4.17.426 sisnan

sisnan returns .TRUE. if its argument is NaN, and .FALSE. otherwise. To be replaced by the Fortran 2003 intrinsic in the future.

Syntax

Fortran specification:

```
use armpl_library

logical function sisnan(SIN)
```

C specification:

```
#include "armpl.h"

armpl_int_t sisnan_(const float *sin);
```

Parameters

SIN Input parameter.

SIN is REAL

Input to test for NaN.

Related Information

For this routine in other precisions, please see *disnan*.

4.17.427 sla_gbamv

SLA_GBAMV performs one of the matrix-vector operations

```
      y := alpha*abs(A)*abs(x) + beta*abs(y),
or    y := alpha*abs(A)**T*abs(x) + beta*abs(y),
```

where alpha and beta are scalars, x and y are vectors and A is an m by n matrix.

This function is primarily used in calculating error bounds. To protect against underflow during evaluation, components in the resulting vector are perturbed away from zero by (N+1) times the underflow threshold. To prevent unnecessarily large errors for block-structure embedded in general matrices, “symbolically” zero components are not perturbed. A zero entry is considered “symbolic” if all multiplications involved in computing that entry have at least one zero multiplicand.

Syntax

Fortran specification:

```
use armpl_library

subroutine sla_gbamv(TRANS, M, N, KL, KU, ALPHA, AB, LDAB, X, INCX, BETA, Y,
                     INCY)
```

C specification:

```
#include "armpl.h"

void sla_gbamv_(const armpl_int_t *trans, const armpl_int_t *m,
                const armpl_int_t *n, const armpl_int_t *kl,
                const armpl_int_t *ku, const float *alpha, const float *ab,
                const armpl_int_t *ldab, const float *x,
                const armpl_int_t *incx, const float *beta, float *y,
                const armpl_int_t *incy);
```

Parameters

TRANS Input parameter.

TRANS is INTEGER

On entry, TRANS specifies the operation to be performed as follows:

BLAS_NO_TRANS y := alpha*abs(A)*abs(x) + beta*abs(y) BLAS_TRANS y := alpha*abs(A^T)*abs(x) + beta*abs(y) BLAS_CONJ_TRANS y := alpha*abs(A^T)*abs(x) + beta*abs(y)

Unchanged on exit.

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of the matrix A. M must be at least zero. Unchanged on exit.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix A. N must be at least zero. Unchanged on exit.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

ALPHA Input parameter.

ALPHA is REAL

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

AB Input parameter.

AB is REAL

AB is an array, dimension (LDAB, n). Before entry, the leading m by n part of the array AB must contain the matrix of coefficients. Unchanged on exit.

LDAB Input parameter.

LDAB is INTEGER

On entry, LDA specifies the first dimension of AB as declared in the calling (sub) program. LDAB must be at least max(1, m). Unchanged on exit.

X Input parameter.

X is REAL

X is an array, dimension. (1 + (n - 1)*abs(INCX)) when TRANS = 'N' or 'n' and at least (1 + (m - 1)*abs(INCX)) otherwise. Before entry, the incremented array X must contain the vector x. Unchanged on exit.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

BETA Input parameter.

BETA is REAL

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input. Unchanged on exit.

Y Input and output parameter.

Y is REAL

Y is an array, dimension. (1 + (m - 1)*abs(INCY)) when TRANS = 'N' or 'n' and at least (1 + (n - 1)*abs(INCY)) otherwise. Before entry with BETA non-zero, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero. Unchanged on exit.

Level 2 Blas routine.

Related Information

For this routine in other precisions, please see *cla_gbamv*, *dla_gbamv* and *zla_gbamv*.

4.17.428 sla_gbrcond

```
SLA_GBRCOND Estimates the Skeel condition number of  op(A) * op2(C)
   where op2 is determined by CMODE as follows
   CMODE =  1    op2(C) = C
   CMODE =  0    op2(C) = I
   CMODE = -1    op2(C) = inv(C)
   The Skeel condition number  cond(A) = norminf( |inv(A)||A| )
   is computed by computing scaling factors R such that
   diag(R)*A*op2(C) is row equilibrated and computing the standard
   infinity-norm condition number.
```

Syntax

Fortran specification:

```
use armpl_library

real function sla_gbrcond(TRANS, N, KL, KU, AB, LDAB, AFB, LDAFB, IPIV, CMODE,
                          C, INFO, WORK, IWORK)
```

C specification:

```
#include "armpl.h"

float sla_gbrcond_(const char *trans, const armpl_int_t *n,
                   const armpl_int_t *kl, const armpl_int_t *ku,
                   const float *ab, const armpl_int_t *ldab, const float *afb,
                   const armpl_int_t *ldafb, const armpl_int_t *ipiv,
                   const armpl_int_t *cmode, const float *c,
                   armpl_int_t *info, float *work, armpl_int_t *iwork, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate Transpose = Transpose)

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

AB Input parameter.

AB is REAL

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(KU+1+i-j,j) = A(i,j) for max(1,j-KU)<=i<=min(N,j+kl)

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KL+KU+1.

AFB Input parameter.

AFB is REAL

AFB is an array, dimension (LDAFB, N). Details of the LU factorization of the band matrix A, as computed by SGBTRF. U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1.

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AFB. LDAFB >= 2*KL+KU+1.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from the factorization A = P*L*U as computed by SGBTRF; row i of the matrix was interchanged with row IPIV(i).

CMODE Input parameter.

CMODE is INTEGER

Determines op2(C) in the formula op(A) * op2(C) as follows: CMODE = 1 op2(C) = C CMODE = 0 op2(C) = I CMODE = -1 op2(C) = inv(C)

C Input parameter.

C is REAL

C is an array, dimension (N). The vector C in the formula op(A) * op2(C).

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. i > 0: The ith argument is invalid.

WORK Input parameter.

WORK is REAL

WORK is an array, dimension (5*N).. Workspace.

IWORK Input parameter.

IWORK is INTEGER array, dimension (N).

Workspace.

Related Information

For this routine in other precisions, please see *dla_gbrcond*.

4.17.429 sla_gbrfsx_extended

sla_gbrfsx_extended improves the computed solution to a system of linear equations by performing extra-precise iterative refinement and provides error bounds and backward error estimates for the solution. This subroutine is called by SGBRFSX to perform iterative refinement. In addition to normwise error bound, the code provides maximum componentwise error bound if possible. See comments for ERR_BNDS_NORM and ERR_BNDS_COMP for details of the error bounds. Note that this subroutine is only resonsible for setting the second fields of ERR_BNDS_NORM and ERR_BNDS_COMP.

Syntax

Fortran specification:

```
use armpl_library

subroutine sla_gbrfsx_extended(PREC_TYPE, TRANS_TYPE, N, KL, KU, NRHS, AB,
                               LDAB, AFB, LDAFB, IPIV, COLEQU, C, B, LDB, Y,
                               LDY, BERR_OUT, N_NORMS, ERR_BNDS_NORM,
                               ERR_BNDS_COMP, RES, AYB, DY, Y_TAIL, RCOND,
                               ITHRESH, RTHRESH, DZ_UB, IGNORE_CWISE, INFO)
```

C specification:

```
#include "armpl.h"

void sla_gbrfsx_extended_(const armpl_int_t *prec_type,
                          const armpl_int_t *trans_type, const armpl_int_t *n,
                          const armpl_int_t *kl, const armpl_int_t *ku,
                          const armpl_int_t *nrhs, const float *ab,
                          const armpl_int_t *ldab, const float *afb,
                          const armpl_int_t *ldafb, const armpl_int_t *ipiv,
                          const armpl_int_t *colequ, const float *c,
                          const float *b, const armpl_int_t *ldb, float *y,
                          const armpl_int_t *ldy, float *berr_out,
                          const armpl_int_t *n_norms, float *err_bnds_norm,
                          float *err_bnds_comp, float *res, float *ayb,
                          float *dy, float *y_tail, const float *rcond,
                          const armpl_int_t *ithresh, const float *rthresh,
                          const float *dz_ub, const armpl_int_t *ignore_cwise,
                          armpl_int_t *info);
```

Parameters

PREC_TYPE Input parameter.

PREC_TYPE is INTEGER

Specifies the intermediate precision to be used in refinement. The value is defined by ILAPREC(P) where P is a CHARACTER and P = 'S': Single = 'D': Double = 'I': Indigenous = 'X', 'E': Extra

TRANS_TYPE Input parameter.

TRANS_TYPE is INTEGER

Specifies the transposition operation on A. The value is defined by ILATRANS(T) where T is a CHARACTER and T = 'N': No transpose = 'T': Transpose = 'C': Conjugate transpose

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0

NRHS Input parameter.

NRHS is INTEGER

The number of right-hand-sides, i.e., the number of columns of the matrix B.

AB Input parameter.

AB is REAL

AB is an array, dimension (LDAB, N). On entry, the N-by-N matrix AB.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= max(1, N).

AFB Input parameter.

AFB is REAL

AFB is an array, dimension (LDAFB, N). The factors L and U from the factorization A = P*L*U as computed by SGBTRF.

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AF. LDAFB >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from the factorization A = P*L*U as computed by SGBTRF; row i of the matrix was interchanged with row IPIV(i).

COLEQU Input parameter.

COLEQU is LOGICAL

If .TRUE. then column equilibration was done to A before calling this routine. This is needed to compute the solution and error bounds correctly.

C Input parameter.

C is REAL

C is an array, dimension (N). The column scale factors for A. If COLEQU = .FALSE., C is not accessed. If C is input, each element of C should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input parameter.

B is REAL

B is an array, dimension (LDB, NRHS). The right-hand-side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Y Input and output parameter.

Y is REAL

Y is an array, dimension (LDY, NRHS). On entry, the solution matrix X, as computed by SGBTRS. On exit, the improved solution matrix Y.

LDY Input parameter.

LDY is INTEGER

The leading dimension of the array Y. LDY >= max(1, N).

BERR_OUT Output parameter.

BERR_OUT is REAL

BERR_OUT is an array, dimension (NRHS). On exit, BERR_OUT(j) contains the componentwise relative backward error for right-hand-side j from the formula max(i) (abs(RES(i)) / (abs(op(A_s))*abs(Y) + abs(B_s))(i)) where abs(Z) is the componentwise absolute value of the matrix or vector Z. This is computed by SLA_LIN_BERR.

N_NORMS Input parameter.

N_NORMS is INTEGER

Determines which error bounds to return (see ERR_BNDS_NORM and ERR_BNDS_COMP). If N_NORMS >= 1 return normwise error bounds. If N_NORMS >= 2 return componentwise error bounds.

ERR_BNDS_NORM Input and output parameter.

ERR_BNDS_NORM is REAL

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

This subroutine is only responsible for setting the second field above. See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Input and output parameter.

ERR_BNDS_COMP is REAL

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:,N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix

Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

This subroutine is only responsible for setting the second field above. See Lapack Working Note 165 for further details and extra cautions.

RES Input parameter.

RES is REAL

RES is an array, dimension (N). Workspace to hold the intermediate residual.

AYB Input parameter.

AYB is REAL

AYB is an array, dimension (N). Workspace. This can be the same workspace passed for Y_TAIL.

DY Input parameter.

DY is REAL

DY is an array, dimension (N). Workspace to hold the intermediate solution.

Y_TAIL Input parameter.

Y_TAIL is REAL

Y_TAIL is an array, dimension (N). Workspace to hold the trailing bits of the intermediate solution.

RCOND Input parameter.

RCOND is REAL

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

ITHRESH Input parameter.

ITHRESH is INTEGER

The maximum number of residual computations allowed for refinement. The default is 10. For 'aggressive' set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in ERR_BNDS_NORM and ERR_BNDS_COMP may no longer be trustworthy.

RTHRESH Input parameter.

RTHRESH is REAL

Determines when to stop refinement if the error estimate stops decreasing. Refinement will stop when the next solution no longer satisfies norm(dx_{i+1}) < RTHRESH * norm(dx_i) where norm(Z) is the infinity norm of Z. RTHRESH satisfies 0 < RTHRESH <= 1. The default value is 0.5. For 'aggressive' set to 0.9 to permit convergence on extremely ill-conditioned matrices. See LAWN 165 for more details.

DZ_UB Input parameter.

DZ_UB is REAL

Determines when to start considering componentwise convergence. Componentwise convergence is only considered after each component of the solution Y is stable, which we definte as the relative change in each component being less than DZ_UB. The default value is 0.25, requiring the first bit to be stable. See LAWN 165 for more details.

IGNORE_CWISE Input parameter.

IGNORE_CWISE is LOGICAL

If .TRUE. then ignore componentwise convergence. Default value is .FALSE..

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. < 0: if INFO = -i, the ith argument to SGBTRS had an illegal value

Related Information

For this routine in other precisions, please see *cla_gbrfsx_extended*, *dla_gbrfsx_extended* and *zla_gbrfsx_extended*.

4.17.430 sla_gbrpvgrw

SLA_GBRPVGRW computes the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If this is much less than 1, the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, estimated condition numbers, and error bounds could be unreliable.

Syntax

Fortran specification:

```
use armpl_library

real function sla_gbrpvgrw(N, KL, KU, NCOLS, AB, LDAB, AFB, LDAFB)
```

C specification:

```
#include "armpl.h"

float sla_gbrpvgrw_(const armpl_int_t *n, const armpl_int_t *kl,
                    const armpl_int_t *ku, const armpl_int_t *ncols,
                    const float *ab, const armpl_int_t *ldab,
                    const float *afb, const armpl_int_t *ldafb);
```

Parameters

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

NCOLS Input parameter.

NCOLS is INTEGER

The number of columns of the matrix A. NCOLS >= 0.

AB Input parameter.

AB is REAL

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(KU+1+i-j,j) = A(i,j) for max(1,j-KU)<=i<=min(N,j+kl)

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KL+KU+1.

AFB Input parameter.

AFB is REAL

AFB is an array, dimension (LDAFB, N). Details of the LU factorization of the band matrix A, as computed by SGBTRF. U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1.

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AFB. LDAFB >= 2*KL+KU+1.

Related Information

For this routine in other precisions, please see *cla_gbrpvgrw*, *dla_gbrpvgrw* and *zla_gbrpvgrw*.

4.17.431 sla_geamv

SLA_GEAMV performs one of the matrix-vector operations

```
     y := alpha*abs(A)*abs(x) + beta*abs(y),
or   y := alpha*abs(A)**T*abs(x) + beta*abs(y),
```

where alpha and beta are scalars, x and y are vectors and A is an m by n matrix.

This function is primarily used in calculating error bounds. To protect against underflow during evaluation, components in the resulting vector are perturbed away from zero by (N+1) times the underflow threshold. To prevent unnecessarily large errors for block-structure embedded in general matrices, "symbolically" zero components are not perturbed. A zero entry is considered "symbolic" if all multiplications involved in computing that entry have at least one zero multiplicand.

Syntax

Fortran specification:

```
use armpl_library

subroutine sla_geamv(TRANS, M, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
```

C specification:

```
#include "armpl.h"

void sla_geamv_(const armpl_int_t *trans, const armpl_int_t *m,
                const armpl_int_t *n, const float *alpha, const float *a,
                const armpl_int_t *lda, const float *x,
                const armpl_int_t *incx, const float *beta, float *y,
                const armpl_int_t *incy);
```

Parameters

TRANS Input parameter.

TRANS is INTEGER

On entry, TRANS specifies the operation to be performed as follows:

BLAS_NO_TRANS y := alpha*abs(A)*abs(x) + beta*abs(y) BLAS_TRANS y := alpha*abs(A^T)*abs(x) + beta*abs(y) BLAS_CONJ_TRANS y := alpha*abs(A^T)*abs(x) + beta*abs(y)

Unchanged on exit.

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of the matrix A. M must be at least zero. Unchanged on exit.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix A. N must be at least zero. Unchanged on exit.

ALPHA Input parameter.

ALPHA is REAL

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

A Input parameter.

A is REAL

A is an array, dimension (LDA, n). Before entry, the leading m by n part of the array A must contain the matrix of coefficients. Unchanged on exit.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, m). Unchanged on exit.

X Input parameter.

X is REAL

X is an array, dimension. (1 + (n - 1)*abs(INCX)) when TRANS = 'N' or 'n' and at least (1 + (m - 1)*abs(INCX)) otherwise. Before entry, the incremented array X must contain the vector x. Unchanged on exit.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

BETA Input parameter.

BETA is REAL

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input. Unchanged on exit.

Y Input and output parameter.

Y is REAL

Y is an array,. dimension at least (1 + (m - 1)*abs(INCY)) when TRANS = 'N' or 'n' and at least (1 + (n - 1)*abs(INCY)) otherwise. Before entry with BETA non-zero, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero. Unchanged on exit.

Level 2 Blas routine.

Related Information

For this routine in other precisions, please see *cla_geamv*, *dla_geamv* and *zla_geamv*.

4.17.432 sla_gercond

```
SLA_GERCOND estimates the Skeel condition number of op(A) * op2(C)
   where op2 is determined by CMODE as follows
   CMODE =  1    op2(C) = C
   CMODE =  0    op2(C) = I
   CMODE = -1    op2(C) = inv(C)
   The Skeel condition number cond(A) = norminf( |inv(A)||A| )
   is computed by computing scaling factors R such that
   diag(R)*A*op2(C) is row equilibrated and computing the standard
   infinity-norm condition number.
```

Syntax

Fortran specification:

```
use armpl_library

real function sla_gercond(TRANS, N, A, LDA, AF, LDAF, IPIV, CMODE, C, INFO,
                          WORK, IWORK)
```

C specification:

```
#include "armpl.h"

float sla_gercond_(const char *trans, const armpl_int_t *n, const float *a,
                   const armpl_int_t *lda, const float *af,
                   const armpl_int_t *ldaf, const armpl_int_t *ipiv,
                   const armpl_int_t *cmode, const float *c,
                   armpl_int_t *info, float *work, armpl_int_t *iwork, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate Transpose = Transpose)

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is REAL

AF is an array, dimension (LDAF, N). The factors L and U from the factorization A = P*L*U as computed by SGETRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from the factorization A = P*L*U as computed by SGETRF; row i of the matrix was interchanged with row IPIV(i).

CMODE Input parameter.

CMODE is INTEGER

Determines op2(C) in the formula op(A) * op2(C) as follows: CMODE = 1 op2(C) = C CMODE = 0 op2(C) = I CMODE = -1 op2(C) = inv(C)

C Input parameter.

C is REAL

C is an array, dimension (N). The vector C in the formula op(A) * op2(C).

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. i > 0: The ith argument is invalid.

WORK Input parameter.

WORK is REAL

WORK is an array, dimension (3*N).. Workspace.

IWORK Input parameter.

IWORK is INTEGER array, dimension (N).

Workspace.2

Related Information

For this routine in other precisions, please see *dla_gercond*.

4.17.433 sla_gerfsx_extended

`sla_gerfsx_extended` improves the computed solution to a system of linear equations by performing extra-precise iterative refinement and provides error bounds and backward error estimates for the solution. This subroutine is called by SGERFSX to perform iterative refinement. In addition to normwise error bound, the code provides maximum componentwise error bound if possible. See comments for ERRS_N and ERRS_C for details of the error bounds. Note that this subroutine is only resonsible for setting the second fields of ERRS_N and ERRS_C.

Syntax

Fortran specification:

```
use armpl_library

subroutine sla_gerfsx_extended(PREC_TYPE, TRANS_TYPE, N, NRHS, A, LDA, AF,
                               LDAF, IPIV, COLEQU, C, B, LDB, Y, LDY,
                               BERR_OUT, N_NORMS, ERRS_N, ERRS_C, RES, AYB,
                               DY, Y_TAIL, RCOND, ITHRESH, RTHRESH, DZ_UB,
                               IGNORE_CWISE, INFO)
```

C specification:

```
#include "armpl.h"

void sla_gerfsx_extended_(const armpl_int_t *prec_type,
                          const armpl_int_t *trans_type, const armpl_int_t *n,
                          const armpl_int_t *nrhs, const float *a,
                          const armpl_int_t *lda, const float *af,
                          const armpl_int_t *ldaf, const armpl_int_t *ipiv,
                          const armpl_int_t *colequ, const float *c,
                          const float *b, const armpl_int_t *ldb, float *y,
                          const armpl_int_t *ldy, float *berr_out,
                          const armpl_int_t *n_norms, float *errs_n,
                          float *errs_c, float *res, float *ayb, float *dy,
                          float *y_tail, const float *rcond,
                          const armpl_int_t *ithresh, const float *rthresh,
                          const float *dz_ub, const armpl_int_t *ignore_cwise,
                          armpl_int_t *info);
```

Parameters

PREC_TYPE Input parameter.

PREC_TYPE is INTEGER

Specifies the intermediate precision to be used in refinement. The value is defined by ILAPREC(P) where P is a CHARACTER and P = ‘S’: Single = ‘D’: Double = ‘I’: Indigenous = ‘X’, ‘E’: Extra

TRANS_TYPE Input parameter.

TRANS_TYPE is INTEGER

Specifies the transposition operation on A. The value is defined by ILATRANS(T) where T is a CHARACTER and T = ‘N’: No transpose = ‘T’: Transpose = ‘C’: Conjugate transpose

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right-hand-sides, i.e., the number of columns of the matrix B.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is REAL

AF is an array, dimension (LDAF, N). The factors L and U from the factorization A = P*L*U as computed by SGETRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from the factorization A = P*L*U as computed by SGETRF; row i of the matrix was interchanged with row IPIV(i).

COLEQU Input parameter.

COLEQU is LOGICAL

If .TRUE. then column equilibration was done to A before calling this routine. This is needed to compute the solution and error bounds correctly.

C Input parameter.

C is REAL

C is an array, dimension (N). The column scale factors for A. If COLEQU = .FALSE., C is not accessed. If C is input, each element of C should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input parameter.

B is REAL

B is an array, dimension (LDB, NRHS). The right-hand-side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Y Input and output parameter.

Y is REAL

Y is an array, dimension (LDY, NRHS). On entry, the solution matrix X, as computed by SGETRS. On exit, the improved solution matrix Y.

LDY Input parameter.

LDY is INTEGER

The leading dimension of the array Y. LDY >= max(1, N).

BERR_OUT Output parameter.

BERR_OUT is REAL

BERR_OUT is an array, dimension (NRHS). On exit, BERR_OUT(j) contains the componentwise relative backward error for right-hand-side j from the formula max(i) (abs(RES(i)) / (abs(op(A_s))*abs(Y) + abs(B_s))(i)) where abs(Z) is the componentwise absolute value of the matrix or vector Z. This is computed by SLA_LIN_BERR.

N_NORMS Input parameter.

N_NORMS is INTEGER

Determines which error bounds to return (see ERRS_N and ERRS_C). If N_NORMS >= 1 return normwise error bounds. If N_NORMS >= 2 return componentwise error bounds.

ERRS_N Input and output parameter.

ERRS_N is REAL

ERRS_N is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERRS_N(i,:) corresponds to the ith right-hand side.

The second index in ERRS_N(:,err) contains the following three fields: err = 1 “Trust/don’t trust” boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch(‘Epsilon’).

err = 2 “Guaranteed” error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch(‘Epsilon’). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch(‘Epsilon’) to determine if the error estimate is “guaranteed”. These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

This subroutine is only responsible for setting the second field above. See Lapack Working Note 165 for further details and extra cautions.

ERRS_C Input and output parameter.

ERRS_C is REAL

ERRS_C is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERRS_C is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:,N_ERR_BNDS) entries are returned.

The first index in ERRS_C(i,:) corresponds to the ith right-hand side.

The second index in ERRS_C(:,err) contains the following three fields: err = 1 “Trust/don’t trust” boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch(‘Epsilon’).

err = 2 “Guaranteed” error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch(‘Epsilon’). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch(‘Epsilon’) to determine if the error estimate is “guaranteed”. These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

This subroutine is only responsible for setting the second field above. See Lapack Working Note 165 for further details and extra cautions.

RES Input parameter.

RES is REAL

RES is an array, dimension (N). Workspace to hold the intermediate residual.

AYB Input parameter.

AYB is REAL

AYB is an array, dimension (N). Workspace. This can be the same workspace passed for Y_TAIL.

DY Input parameter.

DY is REAL

DY is an array, dimension (N). Workspace to hold the intermediate solution.

Y_TAIL Input parameter.

Y_TAIL is REAL

Y_TAIL is an array, dimension (N). Workspace to hold the trailing bits of the intermediate solution.

RCOND Input parameter.

RCOND is REAL

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

ITHRESH Input parameter.

ITHRESH is INTEGER

The maximum number of residual computations allowed for refinement. The default is 10. For ‘aggressive’ set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in ERRS_N and ERRS_C may no longer be trustworthy.

RTHRESH Input parameter.

RTHRESH is REAL

Determines when to stop refinement if the error estimate stops decreasing. Refinement will stop when the next solution no longer satisfies norm(dx_{i+1}) < RTHRESH * norm(dx_i) where norm(Z) is the infinity norm of Z. RTHRESH satisfies 0 < RTHRESH <= 1. The default value is 0.5. For ‘aggressive’ set to 0.9 to permit convergence on extremely ill-conditioned matrices. See LAWN 165 for more details.

DZ_UB Input parameter.

DZ_UB is REAL

Determines when to start considering componentwise convergence. Componentwise convergence is only considered after each component of the solution Y is stable, which we definte as the relative change in each

component being less than DZ_UB. The default value is 0.25, requiring the first bit to be stable. See LAWN 165 for more details.

IGNORE_CWISE Input parameter.

IGNORE_CWISE is LOGICAL

If .TRUE. then ignore componentwise convergence. Default value is .FALSE..

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. < 0: if INFO = -i, the ith argument to SGETRS had an illegal value

Related Information

For this routine in other precisions, please see *cla_gerfsx_extended*, *dla_gerfsx_extended* and *zla_gerfsx_extended*.

4.17.434 sla_gerpvgrw

SLA_GERPVGRW computes the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If this is much less than 1, the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, estimated condition numbers, and error bounds could be unreliable.

Syntax

Fortran specification:

```
use armpl_library

real function sla_gerpvgrw(N, NCOLS, A, LDA, AF, LDAF)
```

C specification:

```
#include "armpl.h"

float sla_gerpvgrw_(const armpl_int_t *n, const armpl_int_t *ncols,
                    const float *a, const armpl_int_t *lda, const float *af,
                    const armpl_int_t *ldaf);
```

Parameters

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NCOLS Input parameter.

NCOLS is INTEGER

The number of columns of the matrix A. NCOLS >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is REAL

AF is an array, dimension (LDAF, N). The factors L and U from the factorization A = P*L*U as computed by SGETRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

Related Information

For this routine in other precisions, please see *cla_gerpvgrw*, *dla_gerpvgrw* and *zla_gerpvgrw*.

4.17.435 sla_lin_berr

```
SLA_LIN_BERR computes componentwise relative backward error from
   the formula
       max(i) ( abs(R(i)) / ( abs(op(A_s))*abs(Y) + abs(B_s) )(i) )
   where abs(Z) is the componentwise absolute value of the matrix
   or vector Z.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine sla_lin_berr(N, NZ, NRHS, RES, AYB, BERR)
```

C specification:

```
#include "armpl.h"

void sla_lin_berr_(const armpl_int_t *n, const armpl_int_t *nz,
                   const armpl_int_t *nrhs, const float *res,
                   const float *ayb, float *berr);
```

Parameters

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NZ Input parameter.

NZ is INTEGER

We add (NZ+1)*SLAMCH('Safe minimum') to R(i) in the numerator to guard against spuriously zero residuals. Default value is N.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices AYB, RES, and BERR. NRHS >= 0.

RES Input parameter.

RES is REAL

RES is an array, dimension (N, NRHS). The residual matrix, i.e., the matrix R in the relative backward error formula above.

AYB Input parameter.

AYB is REAL

AYB is an array, dimension (N, NRHS). The denominator in the relative backward error formula above, i.e., the matrix abs(op(A_s))*abs(Y) + abs(B_s). The matrices A, Y, and B are from iterative refinement (see sla_gerfsx_extended.f).

BERR Output parameter.

BERR is REAL

BERR is an array, dimension (NRHS). The componentwise relative backward error from the formula above.

Related Information

For this routine in other precisions, please see *cla_lin_berr*, *dla_lin_berr* and *zla_lin_berr*.

4.17.436 sla_porcond

```
SLA_PORCOND Estimates the Skeel condition number of  op(A) * op2(C)
   where op2 is determined by CMODE as follows
   CMODE =  1    op2(C) = C
   CMODE =  0    op2(C) = I
   CMODE = -1    op2(C) = inv(C)
   The Skeel condition number  cond(A) = norminf( |inv(A)||A| )
   is computed by computing scaling factors R such that
   diag(R)*A*op2(C) is row equilibrated and computing the standard
   infinity-norm condition number.
```

Syntax

Fortran specification:

```
use armpl_library

real function sla_porcond(UPLO, N, A, LDA, AF, LDAF, CMODE, C, INFO, WORK,
                          IWORK)
```

C specification:

```
#include "armpl.h"

float sla_porcond_(const char *uplo, const armpl_int_t *n, const float *a,
                   const armpl_int_t *lda, const float *af,
                   const armpl_int_t *ldaf, const armpl_int_t *cmode,
                   const float *c, armpl_int_t *info, float *work,
                   armpl_int_t *iwork, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is REAL

AF is an array, dimension (LDAF, N). The triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , as computed by SPOTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

CMODE Input parameter.

CMODE is INTEGER

Determines op2(C) in the formula op(A) * op2(C) as follows: CMODE = 1 op2(C) = C CMODE = 0 op2(C) = I CMODE = -1 op2(C) = inv(C)

C Input parameter.

C is REAL

C is an array, dimension (N). The vector C in the formula op(A) * op2(C).

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. i > 0: The ith argument is invalid.

WORK Input parameter.

WORK is REAL

WORK is an array, dimension (3*N).. Workspace.

IWORK Input parameter.

IWORK is INTEGER array, dimension (N).

Workspace.

Related Information

For this routine in other precisions, please see *dla_porcond*.

4.17.437 sla_porfsx_extended

sla_porfsx_extended improves the computed solution to a system of linear equations by performing extra-precise iterative refinement and provides error bounds and backward error estimates for the solution. This subroutine is called by SPORFSX to perform iterative refinement. In addition to normwise error bound, the code provides maximum componentwise error bound if possible. See comments for ERR_BNDS_NORM and ERR_BNDS_COMP for details of the error bounds. Note that this subroutine is only resonsible for setting the second fields of ERR_BNDS_NORM and ERR_BNDS_COMP.

Syntax

Fortran specification:

```
use armpl_library

subroutine sla_porfsx_extended(PREC_TYPE, UPLO, N, NRHS, A, LDA, AF, LDAF,
                               COLEQU, C, B, LDB, Y, LDY, BERR_OUT, N_NORMS,
                               ERR_BNDS_NORM, ERR_BNDS_COMP, RES, AYB, DY,
                               Y_TAIL, RCOND, ITHRESH, RTHRESH, DZ_UB,
                               IGNORE_CWISE, INFO)
```

C specification:

```
#include "armpl.h"

void sla_porfsx_extended_(const armpl_int_t *prec_type, const char *uplo,
                          const armpl_int_t *n, const armpl_int_t *nrhs,
                          const float *a, const armpl_int_t *lda,
                          const float *af, const armpl_int_t *ldaf,
                          const armpl_int_t *colequ, const float *c,
                          const float *b, const armpl_int_t *ldb, float *y,
                          const armpl_int_t *ldy, float *berr_out,
                          const armpl_int_t *n_norms, float *err_bnds_norm,
                          float *err_bnds_comp, float *res, float *ayb,
                          float *dy, float *y_tail, const float *rcond,
                          const armpl_int_t *ithresh, const float *rthresh,
                          const float *dz_ub, const armpl_int_t *ignore_cwise,
                          armpl_int_t *info, ... );
```

Parameters

PREC_TYPE Input parameter.

PREC_TYPE is INTEGER

Specifies the intermediate precision to be used in refinement. The value is defined by ILAPREC(P) where P is a CHARACTER and P = 'S': Single = 'D': Double = 'I': Indigenous = 'X', 'E': Extra

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right-hand-sides, i.e., the number of columns of the matrix B.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is REAL

AF is an array, dimension (LDAF, N). The triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , as computed by SPOTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

COLEQU Input parameter.

COLEQU is LOGICAL

If .TRUE. then column equilibration was done to A before calling this routine. This is needed to compute the solution and error bounds correctly.

C Input parameter.

C is REAL

C is an array, dimension (N). The column scale factors for A. If COLEQU = .FALSE., C is not accessed. If C is input, each element of C should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input parameter.

B is REAL

B is an array, dimension (LDB, NRHS). The right-hand-side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Y Input and output parameter.

Y is REAL

Y is an array, dimension (LDY, NRHS). On entry, the solution matrix X, as computed by SPOTRS. On exit, the improved solution matrix Y.

LDY Input parameter.

LDY is INTEGER

The leading dimension of the array Y. LDY >= max(1, N).

BERR_OUT Output parameter.

BERR_OUT is REAL

BERR_OUT is an array, dimension (NRHS). On exit, BERR_OUT(j) contains the componentwise relative backward error for right-hand-side j from the formula max(i) (abs(RES(i)) / (abs(op(A_s))*abs(Y) + abs(B_s)

)(i)) where abs(Z) is the componentwise absolute value of the matrix or vector Z. This is computed by SLA_LIN_BERR.

N_NORMS Input parameter.

N_NORMS is INTEGER

Determines which error bounds to return (see ERR_BNDS_NORM and ERR_BNDS_COMP). If N_NORMS >= 1 return normwise error bounds. If N_NORMS >= 2 return componentwise error bounds.

ERR_BNDS_NORM Input and output parameter.

ERR_BNDS_NORM is REAL

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 “Trust/don’t trust” boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch(‘Epsilon’).

err = 2 “Guaranteed” error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch(‘Epsilon’). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch(‘Epsilon’) to determine if the error estimate is “guaranteed”. These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

This subroutine is only responsible for setting the second field above. See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Input and output parameter.

ERR_BNDS_COMP is REAL

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:,N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 “Trust/don’t trust” boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch(‘Epsilon’).

err = 2 “Guaranteed” error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch(‘Epsilon’). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

This subroutine is only responsible for setting the second field above. See Lapack Working Note 165 for further details and extra cautions.

RES Input parameter.

RES is REAL

RES is an array, dimension (N). Workspace to hold the intermediate residual.

AYB Input parameter.

AYB is REAL

AYB is an array, dimension (N). Workspace. This can be the same workspace passed for Y_TAIL.

DY Input parameter.

DY is REAL

DY is an array, dimension (N). Workspace to hold the intermediate solution.

Y_TAIL Input parameter.

Y_TAIL is REAL

Y_TAIL is an array, dimension (N). Workspace to hold the trailing bits of the intermediate solution.

RCOND Input parameter.

RCOND is REAL

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

ITHRESH Input parameter.

ITHRESH is INTEGER

The maximum number of residual computations allowed for refinement. The default is 10. For 'aggressive' set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in ERR_BNDS_NORM and ERR_BNDS_COMP may no longer be trustworthy.

RTHRESH Input parameter.

RTHRESH is REAL

Determines when to stop refinement if the error estimate stops decreasing. Refinement will stop when the next solution no longer satisfies norm(dx_{i+1}) < RTHRESH * norm(dx_i) where norm(Z) is the infinity norm of Z. RTHRESH satisfies 0 < RTHRESH <= 1. The default value is 0.5. For 'aggressive' set to 0.9 to permit convergence on extremely ill-conditioned matrices. See LAWN 165 for more details.

DZ_UB Input parameter.

DZ_UB is REAL

Determines when to start considering componentwise convergence. Componentwise convergence is only considered after each component of the solution Y is stable, which we definte as the relative change in each component being less than DZ_UB. The default value is 0.25, requiring the first bit to be stable. See LAWN 165 for more details.

IGNORE_CWISE Input parameter.

IGNORE_CWISE is LOGICAL

If .TRUE. then ignore componentwise convergence. Default value is .FALSE..

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. < 0: if INFO = -i, the ith argument to SPOTRS had an illegal value

Related Information

For this routine in other precisions, please see *cla_porfsx_extended*, *dla_porfsx_extended* and *zla_porfsx_extended*.

4.17.438 sla_porpvgrw

SLA_PORPVGRW computes the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If this is much less than 1, the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, estimated condition numbers, and error bounds could be unreliable.

Syntax

Fortran specification:

```
use armpl_library

real function sla_porpvgrw(UPLO, NCOLS, A, LDA, AF, LDAF, WORK)
```

C specification:

```
#include "armpl.h"

float sla_porpvgrw_(const char *uplo, const armpl_int_t *ncols,
                    const float *a, const armpl_int_t *lda, const float *af,
                    const armpl_int_t *ldaf, float *work, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

NCOLS Input parameter.

NCOLS is INTEGER

The number of columns of the matrix A. NCOLS >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA,N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1,N).

AF Input parameter.

AF is REAL

AF is an array, dimension (LDAF,N). The triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , as computed by SPOTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1,N).

WORK Input parameter.

WORK is REAL

WORK is an array, dimension (2*N) .

Related Information

For this routine in other precisions, please see *cla_porpvgrw*, *dla_porpvgrw* and *zla_porpvgrw*.

4.17.439 sla_syamv

SLA_SYAMV performs the matrix-vector operation

```
y := alpha*abs(A)*abs(x) + beta*abs(y),
```

where alpha and beta are scalars, x and y are vectors and A is an n by n symmetric matrix.

This function is primarily used in calculating error bounds. To protect against underflow during evaluation, components in the resulting vector are perturbed away from zero by (N+1) times the underflow threshold. To prevent unnecessarily large errors for block-structure embedded in general matrices, "symbolically" zero components are not perturbed. A zero entry is considered "symbolic" if all multiplications involved in computing that entry have at least one zero multiplicand.

Syntax

Fortran specification:

```
use armpl_library

subroutine sla_syamv(UPLO, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
```

C specification:

```
#include "armpl.h"

void sla_syamv_(const armpl_int_t *uplo, const armpl_int_t *n,
                const float *alpha, const float *a, const armpl_int_t *lda,
                const float *x, const armpl_int_t *incx, const float *beta,
                float *y, const armpl_int_t *incy);
```

Parameters

UPLO Input parameter.

UPLO is INTEGER

On entry, UPLO specifies whether the upper or lower triangular part of the array A is to be referenced as follows:

UPLO = BLAS_UPPER Only the upper triangular part of A is to be referenced.

UPLO = BLAS_LOWER Only the lower triangular part of A is to be referenced.

Unchanged on exit.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix A. N must be at least zero. Unchanged on exit.

ALPHA Input parameter.

ALPHA is REAL .

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

A Input parameter.

A is REAL

A is an array, dimension (LDA, n).. Before entry, the leading m by n part of the array A must contain the matrix of coefficients. Unchanged on exit.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, n). Unchanged on exit.

X Input parameter.

X is REAL

X is an array, dimension. (1 + (n - 1)*abs(INCX)) Before entry, the incremented array X must contain the vector x. Unchanged on exit.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

BETA Input parameter.

BETA is REAL .

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input. Unchanged on exit.

Y Input and output parameter.

Y is REAL

Y is an array, dimension. (1 + (n - 1)*abs(INCY)) Before entry with BETA non-zero, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero. Unchanged on exit.

Related Information

For this routine in other precisions, please see *cla_syamv*, *dla_syamv* and *zla_syamv*.

4.17.440 sla_syrcond

```
SLA_SYRCOND estimates the Skeel condition number of  op(A) * op2(C)
   where op2 is determined by CMODE as follows
   CMODE =  1    op2(C) = C
   CMODE =  0    op2(C) = I
   CMODE = -1    op2(C) = inv(C)
   The Skeel condition number cond(A) = norminf( |inv(A)||A| )
   is computed by computing scaling factors R such that
   diag(R)*A*op2(C) is row equilibrated and computing the standard
   infinity-norm condition number.
```

Syntax

Fortran specification:

```
use armpl_library

real function sla_syrcond(UPLO, N, A, LDA, AF, LDAF, IPIV, CMODE, C, INFO,
                          WORK, IWORK)
```

C specification:

```
#include "armpl.h"

float sla_syrcond_(const char *uplo, const armpl_int_t *n, const float *a,
                   const armpl_int_t *lda, const float *af,
                   const armpl_int_t *ldaf, const armpl_int_t *ipiv,
                   const armpl_int_t *cmode, const float *c,
                   armpl_int_t *info, float *work, armpl_int_t *iwork, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is REAL

AF is an array, dimension (LDAF, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by SSYTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by SSYTRF.

CMODE Input parameter.

CMODE is INTEGER

Determines op2(C) in the formula op(A) * op2(C) as follows: CMODE = 1 op2(C) = C CMODE = 0 op2(C) = I CMODE = -1 op2(C) = inv(C)

C Input parameter.

C is REAL

C is an array, dimension (N). The vector C in the formula op(A) * op2(C).

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. i > 0: The ith argument is invalid.

WORK Input parameter.

WORK is REAL

WORK is an array, dimension (3*N).. Workspace.

IWORK Input parameter.

IWORK is INTEGER array, dimension (N).

Workspace.

Related Information

For this routine in other precisions, please see *dla_syrcond*.

4.17.441 sla_syrfsx_extended

sla_syrfsx_extended improves the computed solution to a system of linear equations by performing extra-precise iterative refinement and provides error bounds and backward error estimates for the solution. This subroutine is called by SSYRFSX to perform iterative refinement. In addition to normwise error bound, the code provides maximum componentwise error bound if possible. See comments for ERR_BNDS_NORM and ERR_BNDS_COMP for details of the error bounds. Note that this subroutine is only resonsible for setting the second fields of ERR_BNDS_NORM and ERR_BNDS_COMP.

Syntax

Fortran specification:

```
use armpl_library

subroutine sla_syrfsx_extended(PREC_TYPE, UPLO, N, NRHS, A, LDA, AF, LDAF,
                               IPIV, COLEQU, C, B, LDB, Y, LDY, BERR_OUT,
                               N_NORMS, ERR_BNDS_NORM, ERR_BNDS_COMP, RES,
```

(continues on next page)

(continued from previous page)

```
                                AYB, DY, Y_TAIL, RCOND, ITHRESH, RTHRESH,
                                DZ_UB, IGNORE_CWISE, INFO)
```

C specification:

```
#include "armpl.h"

void sla_syrfsx_extended_(const armpl_int_t *prec_type, const char *uplo,
                          const armpl_int_t *n, const armpl_int_t *nrhs,
                          const float *a, const armpl_int_t *lda,
                          const float *af, const armpl_int_t *ldaf,
                          const armpl_int_t *ipiv, const armpl_int_t *colequ,
                          const float *c, const float *b,
                          const armpl_int_t *ldb, float *y,
                          const armpl_int_t *ldy, float *berr_out,
                          const armpl_int_t *n_norms, float *err_bnds_norm,
                          float *err_bnds_comp, float *res, float *ayb,
                          float *dy, float *y_tail, const float *rcond,
                          const armpl_int_t *ithresh, const float *rthresh,
                          const float *dz_ub, const armpl_int_t *ignore_cwise,
                          armpl_int_t *info, ... );
```

Parameters

PREC_TYPE Input parameter.

PREC_TYPE is INTEGER

Specifies the intermediate precision to be used in refinement. The value is defined by ILAPREC(P) where P is a CHARACTER and P = 'S': Single = 'D': Double = 'I': Indigenous = 'X', 'E': Extra

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right-hand-sides, i.e., the number of columns of the matrix B.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is REAL

AF is an array, dimension (LDAF, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by SSYTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by SSYTRF.

COLEQU Input parameter.

COLEQU is LOGICAL

If .TRUE. then column equilibration was done to A before calling this routine. This is needed to compute the solution and error bounds correctly.

C Input parameter.

C is REAL

C is an array, dimension (N). The column scale factors for A. If COLEQU = .FALSE., C is not accessed. If C is input, each element of C should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input parameter.

B is REAL

B is an array, dimension (LDB, NRHS). The right-hand-side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Y Input and output parameter.

Y is REAL

Y is an array, dimension (LDY, NRHS). On entry, the solution matrix X, as computed by SSYTRS. On exit, the improved solution matrix Y.

LDY Input parameter.

LDY is INTEGER

The leading dimension of the array Y. LDY >= max(1, N).

BERR_OUT Output parameter.

BERR_OUT is REAL

BERR_OUT is an array, dimension (NRHS). On exit, BERR_OUT(j) contains the componentwise relative backward error for right-hand-side j from the formula max(i) (abs(RES(i)) / (abs(op(A_s))*abs(Y) + abs(B_s))(i)) where abs(Z) is the componentwise absolute value of the matrix or vector Z. This is computed by SLA_LIN_BERR.

N_NORMS Input parameter.

N_NORMS is INTEGER

Determines which error bounds to return (see ERR_BNDS_NORM and ERR_BNDS_COMP). If N_NORMS >= 1 return normwise error bounds. If N_NORMS >= 2 return componentwise error bounds.

ERR_BNDS_NORM Input and output parameter.

ERR_BNDS_NORM is REAL

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

This subroutine is only responsible for setting the second field above. See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Input and output parameter.

ERR_BNDS_COMP is REAL

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:,N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

This subroutine is only responsible for setting the second field above. See Lapack Working Note 165 for further details and extra cautions.

RES Input parameter.

RES is REAL

RES is an array, dimension (N). Workspace to hold the intermediate residual.

AYB Input parameter.

AYB is REAL

AYB is an array, dimension (N). Workspace. This can be the same workspace passed for Y_TAIL.

DY Input parameter.

DY is REAL

DY is an array, dimension (N). Workspace to hold the intermediate solution.

Y_TAIL Input parameter.

Y_TAIL is REAL

Y_TAIL is an array, dimension (N). Workspace to hold the trailing bits of the intermediate solution.

RCOND Input parameter.

RCOND is REAL

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

ITHRESH Input parameter.

ITHRESH is INTEGER

The maximum number of residual computations allowed for refinement. The default is 10. For 'aggressive' set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in ERR_BNDS_NORM and ERR_BNDS_COMP may no longer be trustworthy.

RTHRESH Input parameter.

RTHRESH is REAL

Determines when to stop refinement if the error estimate stops decreasing. Refinement will stop when the next solution no longer satisfies norm(dx_{i+1}) < RTHRESH * norm(dx_i) where norm(Z) is the infinity norm of Z. RTHRESH satisfies 0 < RTHRESH <= 1. The default value is 0.5. For 'aggressive' set to 0.9 to permit convergence on extremely ill-conditioned matrices. See LAWN 165 for more details.

DZ_UB Input parameter.

DZ_UB is REAL

Determines when to start considering componentwise convergence. Componentwise convergence is only considered after each component of the solution Y is stable, which we definte as the relative change in each component being less than DZ_UB. The default value is 0.25, requiring the first bit to be stable. See LAWN 165 for more details.

IGNORE_CWISE Input parameter.

IGNORE_CWISE is LOGICAL

If .TRUE. then ignore componentwise convergence. Default value is .FALSE..

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. < 0: if INFO = -i, the ith argument to SLA_SYRFSX_EXTENDED had an illegal value

Related Information

For this routine in other precisions, please see *cla_syrfsx_extended*, *dla_syrfsx_extended* and *zla_syrfsx_extended*.

4.17.442 sla_syrpvgrw

SLA_SYRPVGRW computes the reciprocal pivot growth factor norm(A)/norm(U). The “max absolute element” norm is used. If this is much less than 1, the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, estimated condition numbers, and error bounds could be unreliable.

Syntax

Fortran specification:

```
use armpl_library

real function sla_syrpvgrw(UPLO, N, INFO, A, LDA, AF, LDAF, IPIV, WORK)
```

C specification:

```
#include "armpl.h"

float sla_syrpvgrw_(const char *uplo, const armpl_int_t *n,
                    const armpl_int_t *info, const float *a,
                    const armpl_int_t *lda, const float *af,
                    const armpl_int_t *ldaf, const armpl_int_t *ipiv,
                    float *work, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= ‘U’: Upper triangle of A is stored; = ‘L’: Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

INFO Input parameter.

INFO is INTEGER

The value of INFO returned from SSYTRF, .i.e., the pivot in column INFO is exactly 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is REAL

AF is an array, dimension (LDAF, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by SSYTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by SSYTRF.

WORK Input parameter.

WORK is REAL

WORK is an array, dimension (2*N) .

Related Information

For this routine in other precisions, please see *cla_syrpvgrw*, *dla_syrpvgrw* and *zla_syrpvgrw*.

4.17.443 sla_wwaddw

```
SLA_WWADDW adds a vector W into a doubled-single vector (X, Y).

   This works for all extant IBM's hex and binary floating point
   arithmetics, but not for decimal.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine sla_wwaddw(N, X, Y, W)
```

C specification:

```
#include "armpl.h"

void sla_wwaddw_(const armpl_int_t *n, float *x, float *y, const float *w);
```

Parameters

N Input parameter.

N is INTEGER

The length of vectors X, Y, and W.

X Input and output parameter.

X is REAL

X is an array, dimension (N). The first part of the doubled-single accumulation vector.

Y Input and output parameter.

Y is REAL

Y is an array, dimension (N). The second part of the doubled-single accumulation vector.

W Input parameter.

W is REAL

W is an array, dimension (N). The vector to be added.

Related Information

For this routine in other precisions, please see *cla_wwaddw*, *dla_wwaddw* and *zla_wwaddw*.

4.17.444 slabrd

`slabrd` reduces the first NB rows and columns of a real general m by n matrix A to upper or lower bidiagonal form by an orthogonal transformation Q^T * A * P, and returns the matrices X and Y which are needed to apply the transformation to the unreduced part of A.

If m >= n, A is reduced to upper bidiagonal form; if m < n, to lower bidiagonal form.

This is an auxiliary routine called by SGEBRD

Syntax

Fortran specification:

```
use armpl_library

subroutine slabrd(M, N, NB, A, LDA, D, E, TAUQ, TAUP, X, LDX, Y, LDY)
```

C specification:

```
#include "armpl.h"

void slabrd_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *nb, float *a, const armpl_int_t *lda,
             float *d, float *e, float *tauq, float *taup, float *x,
             const armpl_int_t *ldx, float *y, const armpl_int_t *ldy);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows in the matrix A.

N Input parameter.

N is INTEGER

The number of columns in the matrix A.

NB Input parameter.

NB is INTEGER

The number of leading rows and columns of A to be reduced.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the m by n general matrix to be reduced. On exit, the first NB rows and columns of the matrix are overwritten; the rest of the array is unchanged. If m >= n, elements on and below the diagonal in the first NB columns, with the array TAUQ, represent the orthogonal matrix Q as a product of elementary reflectors; and elements above the diagonal in the first NB rows, with the array TAUP, represent the orthogonal matrix P as a product of elementary reflectors. If m < n, elements below the diagonal in the first NB columns, with the array TAUQ, represent the orthogonal matrix Q as a product of elementary reflectors, and elements on and above the diagonal in the first NB rows, with the array TAUP, represent the orthogonal matrix P as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

D Output parameter.

D is REAL

D is an array, dimension (NB). The diagonal elements of the first NB rows and columns of the reduced matrix. D(i) = A(i,i).

E Output parameter.

E is REAL

E is an array, dimension (NB). The off-diagonal elements of the first NB rows and columns of the reduced matrix.

TAUQ Output parameter.

TAUQ is REAL

TAUQ is an array, dimension (NB). The scalar factors of the elementary reflectors which represent the orthogonal matrix Q. See Further Details.

TAUP Output parameter.

TAUP is REAL

TAUP is an array, dimension (NB). The scalar factors of the elementary reflectors which represent the orthogonal matrix P. See Further Details.

X Output parameter.

X is REAL

X is an array, dimension (LDX, NB). The m-by-nb matrix X required to update the unreduced part of A.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, M).

Y Output parameter.

Y is REAL

Y is an array, dimension (LDY, NB). The n-by-nb matrix Y required to update the unreduced part of A.

LDY Input parameter.

LDY is INTEGER

The leading dimension of the array Y. LDY >= max(1, N).

Related Information

For this routine in other precisions, please see *clabrd*, *dlabrd* and *zlabrd*.

4.17.445 slacn2

slacn2 estimates the 1-norm of a square, real matrix A. Reverse communication is used for evaluating matrix-vector products.

Syntax

Fortran specification:

```
use armpl_library

subroutine slacn2(N, V, X, ISGN, EST, KASE, ISAVE)
```

C specification:

```
#include "armpl.h"

void slacn2_(const armpl_int_t *n, float *v, float *x, armpl_int_t *isgn,
             float *est, armpl_int_t *kase, armpl_int_t *isave);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix. N >= 1.

V Output parameter.

V is REAL

V is an array, dimension (N). On the final return, V = A*W, where EST = norm(V)/norm(W) (W is not returned).

X Input and output parameter.

X is REAL

X is an array, dimension (N). On an intermediate return, X should be overwritten by A * X, if KASE=1, A^T * X, if KASE=2, and SLACN2 must be re-called with all the other parameters unchanged.

ISGN Output parameter.

ISGN is INTEGER array, dimension (N)

EST Input and output parameter.

EST is REAL

On entry with KASE = 1 or 2 and ISAVE(1) = 3, EST should be unchanged from the previous call to SLACN2. On exit, EST is an estimate (a lower bound) for norm(A).

KASE Input and output parameter.

KASE is INTEGER

On the initial call to SLACN2, KASE should be 0. On an intermediate return, KASE will be 1 or 2, indicating whether X should be overwritten by A * X or A^T * X. On the final return from SLACN2, KASE will again be 0.

ISAVE Input and output parameter.

ISAVE is INTEGER array, dimension (3)

ISAVE is used to save variables between calls to SLACN2

Related Information

For this routine in other precisions, please see *clacn2*, *dlacn2* and *zlacn2*. It also exists with a native C interface as *LAPACKE_slacn2*.

4.17.446 slacon

`slacon` estimates the 1-norm of a square, real matrix A. Reverse communication is used for evaluating matrix-vector products.

Syntax

Fortran specification:

```
use armpl_library

subroutine slacon(N, V, X, ISGN, EST, KASE)
```

C specification:

```
#include "armpl.h"

void slacon_(const armpl_int_t *n, float *v, float *x, armpl_int_t *isgn,
             float *est, armpl_int_t *kase);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix. N >= 1.

V Output parameter.

V is REAL

V is an array, dimension (N). On the final return, V = A*W, where EST = norm(V)/norm(W) (W is not returned).

X Input and output parameter.

X is REAL

X is an array, dimension (N). On an intermediate return, X should be overwritten by A * X, if KASE=1, A^T * X, if KASE=2, and SLACON must be re-called with all the other parameters unchanged.

ISGN Output parameter.

ISGN is INTEGER array, dimension (N)

EST Input and output parameter.

EST is REAL

On entry with KASE = 1 or 2 and JUMP = 3, EST should be unchanged from the previous call to SLACON. On exit, EST is an estimate (a lower bound) for norm(A).

KASE Input and output parameter.

KASE is INTEGER

On the initial call to SLACON, KASE should be 0. On an intermediate return, KASE will be 1 or 2, indicating whether X should be overwritten by A * X or A^T * X. On the final return from SLACON, KASE will again be 0.

Related Information

For this routine in other precisions, please see *clacon*, *dlacon* and *zlacon*.

4.17.447 slacpy

`slacpy` copies all or part of a two-dimensional matrix A to another matrix B.

Syntax

Fortran specification:

```
use armpl_library

subroutine slacpy(UPLO, M, N, A, LDA, B, LDB)
```

C specification:

```
#include "armpl.h"

void slacpy_(const char *uplo, const armpl_int_t *m, const armpl_int_t *n,
             const float *a, const armpl_int_t *lda, float *b,
             const armpl_int_t *ldb, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies the part of the matrix A to be copied to B. = 'U': Upper triangular part = 'L': Lower triangular part Otherwise: All of the matrix A

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). The m by n matrix A. If UPLO = 'U', only the upper triangle or trapezoid is accessed; if UPLO = 'L', only the lower triangle or trapezoid is accessed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Output parameter.

B is REAL

B is an array, dimension (LDB, N). On exit, B = A in the locations specified by UPLO.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M).

Related Information

For this routine in other precisions, please see *clacpy*, *dlacpy* and *zlacpy*. It also exists with a native C interface as *LAPACKE_slacpy*.

4.17.448 sladiv

`sladiv` performs complex division in real arithmetic

```
            a + i*b
p + i*q = ---------
            c + i*d
```

The algorithm is due to Michael Baudin and Robert L. Smith and can be found in the paper "A Robust Complex Division in Scilab"

Syntax

Fortran specification:

```
use armpl_library

subroutine sladiv(A, B, C, D, P, Q)
```

C specification:

```
#include "armpl.h"

void sladiv_(const float *a, const float *b, const float *c, const float *d,
             float *p, float *q);
```

Parameters

A Input parameter.

A is REAL

B Input parameter.

B is REAL

C Input parameter.

C is REAL

D Input parameter.

D is REAL

The scalars a, b, c, and d in the above expression.

P Output parameter.

P is REAL

Q Output parameter.

Q is REAL

The scalars p and q in the above expression.

Related Information

For this routine in other precisions, please see *cladiv*, *dladiv* and *zladiv*.

4.17.449 slae2

`slae2` computes the eigenvalues of a 2-by-2 symmetric matrix

```
[  A   B  ]
[  B   C  ].
```

On return, RT1 is the eigenvalue of larger absolute value, and RT2 is the eigenvalue of smaller absolute value.

Syntax

Fortran specification:

```
use armpl_library

subroutine slae2(A, B, C, RT1, RT2)
```

C specification:

```
#include "armpl.h"

void slae2_(const float *a, const float *b, const float *c, float *rt1,
            float *rt2);
```

Parameters

A Input parameter.

A is REAL

The (1,1) element of the 2-by-2 matrix.

B Input parameter.

B is REAL

The (1,2) and (2,1) elements of the 2-by-2 matrix.

C Input parameter.

C is REAL

The (2,2) element of the 2-by-2 matrix.

RT1 Output parameter.

RT1 is REAL

The eigenvalue of larger absolute value.

RT2 Output parameter.

RT2 is REAL

The eigenvalue of smaller absolute value.

Related Information

For this routine in other precisions, please see *dlae2*.

4.17.450 slaebz

`slaebz` contains the iteration loops which compute and use the function N(w), which is the count of eigenvalues of a symmetric tridiagonal matrix T less than or equal to its argument w. It performs a choice of two types of loops:

IJOB=1, followed by IJOB=2: It takes as input a list of intervals and returns a list of

```
sufficiently small intervals whose union contains the same
eigenvalues as the union of the original intervals.
The input intervals are (AB(j,1),AB(j,2)], j=1,...,MINP.
The output interval (AB(j,1),AB(j,2)] will contain
eigenvalues NAB(j,1)+1,...,NAB(j,2), where 1 <= j <= MOUT.
```

IJOB=3: It performs a binary search in each input interval

```
(AB(j,1),AB(j,2)] for a point  w(j)  such that
N(w(j))=NVAL(j), and uses  C(j)  as the starting point of
the search.  If such a w(j) is found, then on output
AB(j,1)=AB(j,2)=w.  If no such w(j) is found, then on output
(AB(j,1),AB(j,2)] will be a small interval containing the
point where N(w) jumps through NVAL(j), unless that point
lies outside the initial interval.
```

Note that the intervals are in all cases half-open intervals, i.e., of the form (a,b] , which includes b but not a .

To avoid underflow, the matrix should be scaled so that its largest element is no greater than overflow**(1/2) * underflow**(1/4) in absolute value. To assure the most accurate computation of small eigenvalues, the matrix should be scaled to be not much smaller than that, either.

See W. Kahan “Accurate Eigenvalues of a Symmetric Tridiagonal Matrix”, Report CS41, Computer Science Dept., Stanford University, July 21, 1966

Note: the arguments are, in general, *not* checked for unreasonable values.

Syntax

Fortran specification:

```
use armpl_library

subroutine slaebz(IJOB, NITMAX, N, MMAX, MINP, NBMIN, ABSTOL, RELTOL, PIVMIN,
                  D, E, E2, NVAL, AB, C, MOUT, NAB, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void slaebz_(const armpl_int_t *ijob, const armpl_int_t *nitmax,
             const armpl_int_t *n, const armpl_int_t *mmax,
             const armpl_int_t *minp, const armpl_int_t *nbmin,
             const float *abstol, const float *reltol, const float *pivmin,
             const float *d, const float *e, const float *e2,
             armpl_int_t *nval, float *ab, float *c, armpl_int_t *mout,
             armpl_int_t *nab, float *work, armpl_int_t *iwork,
             armpl_int_t *info);
```

Parameters

IJOB Input parameter.

IJOB is INTEGER

Specifies what is to be done: = 1: Compute NAB for the initial intervals. = 2: Perform bisection iteration to find eigenvalues of T. = 3: Perform bisection iteration to invert N(w), i.e., to find a point which has a specified number of eigenvalues of T to its left. Other values will cause SLAEBZ to return with INFO=-1.

NITMAX Input parameter.

NITMAX is INTEGER

The maximum number of "levels" of bisection to be performed, i.e., an interval of width W will not be made smaller than 2^(-NITMAX) * W. If not all intervals have converged after NITMAX iterations, then INFO is set to the number of non-converged intervals.

N Input parameter.

N is INTEGER

The dimension n of the tridiagonal matrix T. It must be at least 1.

MMAX Input parameter.

MMAX is INTEGER

The maximum number of intervals. If more than MMAX intervals are generated, then SLAEBZ will quit with INFO=MMAX+1.

MINP Input parameter.

MINP is INTEGER

The initial number of intervals. It may not be greater than MMAX.

NBMIN Input parameter.

NBMIN is INTEGER

The smallest number of intervals that should be processed using a vector loop. If zero, then only the scalar loop will be used.

ABSTOL Input parameter.

ABSTOL is REAL

The minimum (absolute) width of an interval. When an interval is narrower than ABSTOL, or than RELTOL times the larger (in magnitude) endpoint, then it is considered to be sufficiently small, i.e., converged. This must be at least zero.

RELTOL Input parameter.

RELTOL is REAL

The minimum relative width of an interval. When an interval is narrower than ABSTOL, or than RELTOL times the larger (in magnitude) endpoint, then it is considered to be sufficiently small, i.e., converged. Note: this should always be at least radix*machine epsilon.

PIVMIN Input parameter.

PIVMIN is REAL

The minimum absolute value of a "pivot" in the Sturm sequence loop. This must be at least max |e(j)**2|*safe_min and at least safe_min, where safe_min is at least the smallest number that can divide one without overflow.

D Input parameter.

D is REAL

D is an array, dimension (N). The diagonal elements of the tridiagonal matrix T.

E Input parameter.

E is REAL

E is an array, dimension (N). The offdiagonal elements of the tridiagonal matrix T in positions 1 through N-1. E(N) is arbitrary.

E2 Input parameter.

E2 is REAL

E2 is an array, dimension (N). The squares of the offdiagonal elements of the tridiagonal matrix T. E2(N) is ignored.

NVAL Input and output parameter.

NVAL is INTEGER array, dimension (MINP)

If IJOB=1 or 2, not referenced. If IJOB=3, the desired values of N(w). The elements of NVAL will be reordered to correspond with the intervals in AB. Thus, NVAL(j) on output will not, in general be the same as NVAL(j) on input, but it will correspond with the interval (AB(j,1),AB(j,2)] on output.

AB Input and output parameter.

AB is REAL

AB is an array, dimension (MMAX,2). The endpoints of the intervals. AB(j,1) is a(j), the left endpoint of the j-th interval, and AB(j,2) is b(j), the right endpoint of the j-th interval. The input intervals will, in general, be modified, split, and reordered by the calculation.

C Input and output parameter.

C is REAL

C is an array, dimension (MMAX). If IJOB=1, ignored. If IJOB=2, workspace. If IJOB=3, then on input C(j) should be initialized to the first search point in the binary search.

MOUT Output parameter.

MOUT is INTEGER

If IJOB=1, the number of eigenvalues in the intervals. If IJOB=2 or 3, the number of intervals output. If IJOB=3, MOUT will equal MINP.

NAB Input and output parameter.

NAB is INTEGER array, dimension (MMAX,2)

If IJOB=1, then on output NAB(i,j) will be set to N(AB(i,j)). If IJOB=2, then on input, NAB(i,j) should be set. It must satisfy the condition: N(AB(i,1)) <= NAB(i,1) <= NAB(i,2) <= N(AB(i,2)), which means that in interval i only eigenvalues NAB(i,1)+1,...,NAB(i,2) will be considered. Usually, NAB(i,j)=N(AB(i,j)), from a previous call to SLAEBZ with IJOB=1. On output, NAB(i,j) will contain max(na(k),min(nb(k),N(AB(i,j)))), where k is the index of the input interval that the output interval (AB(j,1),AB(j,2)] came from, and na(k) and nb(k) are the the input values of NAB(k,1) and NAB(k,2). If IJOB=3, then on output, NAB(i,j) contains N(AB(i,j)), unless N(w) > NVAL(i) for all search points w , in which case NAB(i,1) will not be modified, i.e., the output value will be the same as the input value (modulo reorderings – see NVAL and AB), or unless N(w) < NVAL(i) for all search points w , in which case NAB(i,2) will not be modified. Normally, NAB should be set to some distinctive value(s) before SLAEBZ is called.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MMAX). Workspace.

IWORK Output parameter.

IWORK is INTEGER array, dimension (MMAX)

Workspace.

INFO Output parameter.

INFO is INTEGER

= 0: All intervals converged. = 1–MMAX: The last INFO intervals did not converge. = MMAX+1: More than MMAX intervals were generated.

Related Information

For this routine in other precisions, please see *dlaebz*.

4.17.451 slaed0

slaed0 computes all eigenvalues and corresponding eigenvectors of a symmetric tridiagonal matrix using the divide and conquer method.

Syntax

Fortran specification:

```
use armpl_library

subroutine slaed0(ICOMPQ, QSIZ, N, D, E, Q, LDQ, QSTORE, LDQS, WORK, IWORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void slaed0_(const armpl_int_t *icompq, const armpl_int_t *qsiz,
             const armpl_int_t *n, float *d, const float *e, float *q,
             const armpl_int_t *ldq, float *qstore, const armpl_int_t *ldqs,
             float *work, armpl_int_t *iwork, armpl_int_t *info);
```

Parameters

ICOMPQ Input parameter.

ICOMPQ is INTEGER

= 0: Compute eigenvalues only. = 1: Compute eigenvectors of original dense symmetric matrix also. On entry, Q contains the orthogonal matrix used to reduce the original matrix to tridiagonal form. = 2: Compute eigenvalues and eigenvectors of tridiagonal matrix.

QSIZ Input parameter.

QSIZ is INTEGER

The dimension of the orthogonal matrix used to reduce the full matrix to tridiagonal form. QSIZ >= N if ICOMPQ = 1.

N Input parameter.

N is INTEGER

The dimension of the symmetric tridiagonal matrix. N >= 0.

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry, the main diagonal of the tridiagonal matrix. On exit, its eigenvalues.

E Input parameter.

E is REAL

E is an array, dimension (N-1). The off-diagonal elements of the tridiagonal matrix. On exit, E has been destroyed.

Q Input and output parameter.

Q is REAL

Q is an array, dimension (LDQ, N). On entry, Q must contain an N-by-N orthogonal matrix. If ICOMPQ = 0 Q is not referenced. If ICOMPQ = 1 On entry, Q is a subset of the columns of the orthogonal matrix used to reduce the full matrix to tridiagonal form corresponding to the subset of the full matrix which is being decomposed at this time. If ICOMPQ = 2 On entry, Q will be the identity matrix. On exit, Q contains the eigenvectors of the tridiagonal matrix.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. If eigenvectors are desired, then LDQ >= max(1, N). In any case, LDQ >= 1.

QSTORE Output parameter.

QSTORE is REAL

QSTORE is an array, dimension (LDQS, N). Referenced only when ICOMPQ = 1. Used to store parts of the eigenvector matrix when the updating matrix multiplies take place.

LDQS Input parameter.

LDQS is INTEGER

The leading dimension of the array QSTORE. If ICOMPQ = 1, then LDQS >= max(1, N). In any case, LDQS >= 1.

WORK Output parameter.

WORK is REAL

WORK is an array,. If ICOMPQ = 0 or 1, the dimension of WORK must be at least 1 + 3* N + 2*N*lg N + 3*N**2 (lg(N) = smallest integer k such that 2^k >= N) If ICOMPQ = 2, the dimension of WORK must be at least 4* N + N**2.

IWORK Output parameter.

IWORK is INTEGER array,

If ICOMPQ = 0 or 1, the dimension of IWORK must be at least 6 + 6* N + 5*N*lg N. (lg(N) = smallest integer k such that 2^k >= N) If ICOMPQ = 2, the dimension of IWORK must be at least 3 + 5*N.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: The algorithm failed to compute an eigenvalue while working on the submatrix lying in rows and columns INFO/(N+1) through mod(INFO,N+1).

Related Information

For this routine in other precisions, please see *claed0*, *dlaed0* and *zlaed0*.

4.17.452 slaed1

slaed1 computes the updated eigensystem of a diagonal matrix after modification by a rank-one symmetric matrix. This routine is used only for the eigenproblem which requires all eigenvalues and eigenvectors of a tridiagonal matrix. SLAED7 handles the case in which eigenvalues only or eigenvalues and eigenvectors of a full symmetric matrix (which was reduced to tridiagonal form) are desired.

```
T = Q(in) ( D(in) + RHO * Z*Z**T ) Q**T(in) = Q(out) * D(out) * Q**T(out)

where Z = Q**T*u, u is a vector of length N with ones in the
CUTPNT and CUTPNT + 1 th elements and zeros elsewhere.

The eigenvectors of the original matrix are stored in Q, and the
eigenvalues are in D.  The algorithm consists of three stages:

   The first stage consists of deflating the size of the problem
   when there are multiple eigenvalues or if there is a zero in
   the Z vector.  For each such occurrence the dimension of the
   secular equation problem is reduced by one.  This stage is
   performed by the routine SLAED2.

   The second stage consists of calculating the updated
   eigenvalues. This is done by finding the roots of the secular
   equation via the routine SLAED4 (as called by SLAED3).
   This routine also calculates the eigenvectors of the current
   problem.

   The final stage consists of computing the updated eigenvectors
   directly using the updated eigenvalues.  The eigenvectors for
   the current problem are multiplied with the eigenvectors from
   the overall problem.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine slaed1(N, D, Q, LDQ, INDXQ, RHO, CUTPNT, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void slaed1_(const armpl_int_t *n, float *d, float *q, const armpl_int_t *ldq,
             armpl_int_t *indxq, const float *rho, const armpl_int_t *cutpnt,
             float *work, armpl_int_t *iwork, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The dimension of the symmetric tridiagonal matrix. N >= 0.

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry, the eigenvalues of the rank-1-perturbed matrix. On exit, the eigenvalues of the repaired matrix.

Q Input and output parameter.

Q is REAL

Q is an array, dimension (LDQ, N). On entry, the eigenvectors of the rank-1-perturbed matrix. On exit, the eigenvectors of the repaired tridiagonal matrix.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, N).

INDXQ Input and output parameter.

INDXQ is INTEGER array, dimension (N)

On entry, the permutation which separately sorts the two subproblems in D into ascending order. On exit, the permutation which will reintegrate the subproblems back into sorted order, i.e. D(INDXQ(I = 1, N)) will be in ascending order.

RHO Input parameter.

RHO is REAL

The subdiagonal entry used to create the rank-1 modification.

CUTPNT Input parameter.

CUTPNT is INTEGER

The location of the last eigenvalue in the leading sub-matrix. min(1, N) <= CUTPNT <= N/2.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (4* N + N2)** .

IWORK Output parameter.

IWORK is INTEGER array, dimension (4*N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = 1, an eigenvalue did not converge

Related Information

For this routine in other precisions, please see *dlaed1*.

4.17.453 slaed2

slaed2 merges the two sets of eigenvalues together into a single sorted set. Then it tries to deflate the size of the problem. There are two ways in which deflation can occur: when two or more eigenvalues are close together or if there is a tiny entry in the Z vector. For each such occurrence the order of the related secular equation problem is reduced by one.

Syntax

Fortran specification:

```
use armpl_library

subroutine slaed2(K, N, N1, D, Q, LDQ, INDXQ, RHO, Z, DLAMDA, W, Q2, INDX,
                  INDXC, INDXP, COLTYP, INFO)
```

C specification:

```
#include "armpl.h"

void slaed2_(armpl_int_t *k, const armpl_int_t *n, const armpl_int_t *n1,
             float *d, float *q, const armpl_int_t *ldq, armpl_int_t *indxq,
             float *rho, const float *z, float *dlamda, float *w, float *q2,
             armpl_int_t *indx, armpl_int_t *indxc, armpl_int_t *indxp,
             armpl_int_t *coltyp, armpl_int_t *info);
```

Parameters

K Output parameter.

K is INTEGER

The number of non-deflated eigenvalues, and the order of the related secular equation. 0 <= K <=N.

N Input parameter.

N is INTEGER

The dimension of the symmetric tridiagonal matrix. N >= 0.

N1 Input parameter.

N1 is INTEGER

The location of the last eigenvalue in the leading sub-matrix. min(1, N) <= N1 <= N/2.

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry, D contains the eigenvalues of the two submatrices to be combined. On exit, D contains the trailing (N-K) updated eigenvalues (those which were deflated) sorted into increasing order.

Q Input and output parameter.

Q is REAL

Q is an array, dimension (LDQ, N). On entry, Q contains the eigenvectors of two submatrices in the two square blocks with corners at (1,1), (N1, N1) and (N1+1, N1+1), (N, N). On exit, Q contains the trailing (N-K) updated eigenvectors (those which were deflated) in its last N-K columns.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, N).

INDXQ Input and output parameter.

INDXQ is INTEGER array, dimension (N)

The permutation which separately sorts the two sub-problems in D into ascending order. Note that elements in the second half of this permutation must first have N1 added to their values. Destroyed on exit.

RHO Input and output parameter.

RHO is REAL

On entry, the off-diagonal element associated with the rank-1 cut which originally split the two submatrices which are now being recombined. On exit, RHO has been modified to the value required by SLAED3.

Z Input parameter.

Z is REAL

Z is an array, dimension (N). On entry, Z contains the updating vector (the last row of the first sub-eigenvector matrix and the first row of the second sub-eigenvector matrix). On exit, the contents of Z have been destroyed by the updating process.

DLAMDA Output parameter.

DLAMDA is REAL

DLAMDA is an array, dimension (N). A copy of the first K eigenvalues which will be used by SLAED3 to form the secular equation.

W Output parameter.

W is REAL

W is an array, dimension (N). The first k values of the final deflation-altered z-vector which will be passed to SLAED3.

Q2 Output parameter.

Q2 is REAL

Q2 is an array, dimension (N1**2+(N-N1)**2). A copy of the first K eigenvectors which will be used by SLAED3 in a matrix multiply (SGEMM) to solve for the new eigenvectors.

INDX Output parameter.

INDX is INTEGER array, dimension (N)

The permutation used to sort the contents of DLAMDA into ascending order.

INDXC Output parameter.

INDXC is INTEGER array, dimension (N)

The permutation used to arrange the columns of the deflated Q matrix into three groups: the first group contains non-zero elements only at and above N1, the second contains non-zero elements only below N1, and the third is dense.

INDXP Output parameter.

INDXP is INTEGER array, dimension (N)

The permutation used to place deflated values of D at the end of the array. INDXP(1:K) points to the nondeflated D-values and INDXP(K+1:N) points to the deflated eigenvalues.

COLTYP Output parameter.

COLTYP is INTEGER array, dimension (N)

During execution, a label which will indicate which of the following types a column in the Q2 matrix is: 1 : non-zero in the upper half only; 2 : dense; 3 : non-zero in the lower half only; 4 : deflated. On exit, COLTYP(i) is the number of columns of type i, for i=1 to 4 only.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *dlaed2*.

4.17.454 slaed3

`slaed3` finds the roots of the secular equation, as defined by the values in D, W, and RHO, between 1 and K. It makes the appropriate calls to SLAED4 and then updates the eigenvectors by multiplying the matrix of eigenvectors of the pair of eigensystems being combined by the matrix of eigenvectors of the K-by-K system which is solved here.

This code makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine slaed3(K, N, N1, D, Q, LDQ, RHO, DLAMDA, Q2, INDX, CTOT, W, S,
                  INFO)
```

C specification:

```
#include "armpl.h"

void slaed3_(const armpl_int_t *k, const armpl_int_t *n,
             const armpl_int_t *n1, float *d, float *q,
             const armpl_int_t *ldq, const float *rho, float *dlamda,
             const float *q2, const armpl_int_t *indx,
             const armpl_int_t *ctot, float *w, float *s, armpl_int_t *info);
```

Parameters

K Input parameter.

`K` is `INTEGER`

The number of terms in the rational function to be solved by SLAED4. `K` >= 0.

N Input parameter.

`N` is `INTEGER`

The number of rows and columns in the `Q` matrix. `N` >= `K` (deflation may result in N>K).

N1 Input parameter.

`N1` is `INTEGER`

The location of the last eigenvalue in the leading submatrix. min(1, `N`) <= `N1` <= N/2.

D Output parameter.

`D` is `REAL`

`D` is an array, dimension (`N`). D(I) contains the updated eigenvalues for 1 <= I <= `K`.

Q Output parameter.

`Q` is `REAL`

`Q` is an array, dimension (`LDQ`, `N`). Initially the first `K` columns are used as workspace. On output the columns 1 to `K` contain the updated eigenvectors.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, N).

RHO Input parameter.

RHO is REAL

The value of the parameter in the rank one update equation. RHO >= 0 required.

DLAMDA Input and output parameter.

DLAMDA is REAL

DLAMDA is an array, dimension (K). The first K elements of this array contain the old roots of the deflated updating problem. These are the poles of the secular equation. May be changed on output by having lowest order bit set to zero on Cray X-MP, Cray Y-MP, Cray-2, or Cray C-90, as described above.

Q2 Input parameter.

Q2 is REAL

Q2 is an array, dimension (LDQ2*N). The first K columns of this matrix contain the non-deflated eigenvectors for the split problem.

INDX Input parameter.

INDX is INTEGER array, dimension (N)

The permutation used to arrange the columns of the deflated Q matrix into three groups (see SLAED2). The rows of the eigenvectors found by SLAED4 must be likewise permuted before the matrix multiply can take place.

CTOT Input parameter.

CTOT is INTEGER array, dimension (4)

A count of the total number of the various types of columns in Q, as described in INDX. The fourth column type is any column which has been deflated.

W Input and output parameter.

W is REAL

W is an array, dimension (K). The first K elements of this array contain the components of the deflation-adjusted updating vector. Destroyed on output.

S Output parameter.

S is REAL

S is an array, dimension (N1 + 1)*K. Will contain the eigenvectors of the repaired matrix which will be multiplied by the previously accumulated eigenvectors to update the system.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = 1, an eigenvalue did not converge

Related Information

For this routine in other precisions, please see *dlaed3*.

4.17.455 slaed4

This subroutine computes the I-th updated eigenvalue of a symmetric rank-one modification to a diagonal matrix whose elements are given in the array d, and that

```
D(i) < D(j)  for  i < j
```

and that RHO > 0. This is arranged by the calling routine, and is no loss in generality. The rank-one modified system is thus

```
diag( D )  +  RHO * Z * Z_transpose.
```

where we assume the Euclidean norm of Z is 1.

The method consists of approximating the rational functions in the secular equation by simpler interpolating rational functions.

Syntax

Fortran specification:

```
use armpl_library

subroutine slaed4(N, I, D, Z, DELTA, RHO, DLAM, INFO)
```

C specification:

```
#include "armpl.h"

void slaed4_(const armpl_int_t *n, const armpl_int_t *i, const float *d,
             const float *z, float *delta, const float *rho, float *dlam,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The length of all arrays.

I Input parameter.

I is INTEGER

The index of the eigenvalue to be computed. 1 <= I <= N.

D Input parameter.

D is REAL

D is an array, dimension (N). The original eigenvalues. It is assumed that they are in order, D(I) < D(J) for I < J.

Z Input parameter.

Z is REAL

Z is an array, dimension (N). The components of the updating vector.

DELTA Output parameter.

DELTA is REAL

DELTA is an array, dimension (N). If N .GT. 2, DELTA contains (D(j) - lambda_I) in its j-th component. If N = 1, then DELTA(1) = 1. If N = 2, see SLAED5 for detail. The vector DELTA contains the information necessary to construct the eigenvectors by SLAED3 and SLAED9.

RHO Input parameter.

RHO is REAL

The scalar in the symmetric updating formula.

DLAM Output parameter.

DLAM is REAL

The computed lambda_I, the I-th updated eigenvalue.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit > 0: if INFO = 1, the updating process failed.

Related Information

For this routine in other precisions, please see *dlaed4*.

4.17.456 slaed5

This subroutine computes the I-th eigenvalue of a symmetric rank-one modification of a 2-by-2 diagonal matrix

```
diag( D )  +  RHO * Z * transpose(Z) .
```

The diagonal elements in the array D are assumed to satisfy

```
D(i) < D(j)  for  i < j .
```

We also assume RHO > 0 and that the Euclidean norm of the vector Z is one.

Syntax

Fortran specification:

```
use armpl_library

subroutine slaed5(I, D, Z, DELTA, RHO, DLAM)
```

C specification:

```
#include "armpl.h"

void slaed5_(const armpl_int_t *i, const float *d, const float *z,
             float *delta, const float *rho, float *dlam);
```

Parameters

I Input parameter.

I is INTEGER

The index of the eigenvalue to be computed. I = 1 or I = 2.

D Input parameter.

D is REAL

D is an array, dimension (2). The original eigenvalues. We assume D(1) < D(2).

Z Input parameter.

Z is REAL

Z is an array, dimension (2). The components of the updating vector.

DELTA Output parameter.

DELTA is REAL

DELTA is an array, dimension (2). The vector DELTA contains the information necessary to construct the eigenvectors.

RHO Input parameter.

RHO is REAL

The scalar in the symmetric updating formula.

DLAM Output parameter.

DLAM is REAL

The computed lambda_I, the I-th updated eigenvalue.

Related Information

For this routine in other precisions, please see *dlaed5*.

4.17.457 slaed6

slaed6 computes the positive or negative root (closest to the origin) of

```
z(1)        z(2)        z(3)
```

f(x) = rho + ——— + ———- + ———

```
d(1)-x      d(2)-x      d(3)-x
```

It is assumed that

```
 if ORGATI = .true. the root is between d(2) and d(3);
otherwise it is between d(1) and d(2)
```

This routine will be called by SLAED4 when necessary. In most cases, the root sought is the smallest in magnitude, though it might not be in some extremely rare situations.

Syntax

Fortran specification:

```
use armpl_library

subroutine slaed6(KNITER, ORGATI, RHO, D, Z, FINIT, TAU, INFO)
```

C specification:

```
#include "armpl.h"

void slaed6_(const armpl_int_t *kniter, const armpl_int_t *orgati,
             const float *rho, const float *d, const float *z,
             const float *finit, float *tau, armpl_int_t *info);
```

Parameters

KNITER Input parameter.

KNITER is INTEGER

Refer to SLAED4 for its significance.

ORGATI Input parameter.

ORGATI is LOGICAL

If ORGATI is true, the needed root is between d(2) and d(3); otherwise it is between d(1) and d(2). See SLAED4 for further details.

RHO Input parameter.

RHO is REAL

Refer to the equation f(x) above.

D Input parameter.

D is REAL

D is an array, dimension (3). D satisfies d(1) < d(2) < d(3).

Z Input parameter.

Z is REAL

Z is an array, dimension (3). Each of the elements in z must be positive.

FINIT Input parameter.

FINIT is REAL

The value of f at 0. It is more accurate than the one evaluated inside this routine (if someone wants to do so).

TAU Output parameter.

TAU is REAL

The root of the equation f(x).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit > 0: if INFO = 1, failure to converge

Related Information

For this routine in other precisions, please see *dlaed6*.

4.17.458 slaed7

slaed7 computes the updated eigensystem of a diagonal matrix after modification by a rank-one symmetric matrix. This routine is used only for the eigenproblem which requires all eigenvalues and optionally eigenvectors of a dense symmetric matrix that has been reduced to tridiagonal form. SLAED1 handles the case in which all eigenvalues and eigenvectors of a symmetric tridiagonal matrix are desired.

```
T = Q(in) ( D(in) + RHO * Z*Z**T ) Q**T(in) = Q(out) * D(out) * Q**T(out)

where Z = Q**Tu, u is a vector of length N with ones in the
CUTPNT and CUTPNT + 1 th elements and zeros elsewhere.

The eigenvectors of the original matrix are stored in Q, and the
eigenvalues are in D.  The algorithm consists of three stages:

   The first stage consists of deflating the size of the problem
   when there are multiple eigenvalues or if there is a zero in
   the Z vector.  For each such occurrence the dimension of the
   secular equation problem is reduced by one.  This stage is
   performed by the routine SLAED8.

   The second stage consists of calculating the updated
   eigenvalues. This is done by finding the roots of the secular
   equation via the routine SLAED4 (as called by SLAED9).
   This routine also calculates the eigenvectors of the current
   problem.

   The final stage consists of computing the updated eigenvectors
   directly using the updated eigenvalues.  The eigenvectors for
   the current problem are multiplied with the eigenvectors from
   the overall problem.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine slaed7(ICOMPQ, N, QSIZ, TLVLS, CURLVL, CURPBM, D, Q, LDQ, INDXQ,
                  RHO, CUTPNT, QSTORE, QPTR, PRMPTR, PERM, GIVPTR, GIVCOL,
                  GIVNUM, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void slaed7_(const armpl_int_t *icompq, const armpl_int_t *n,
             const armpl_int_t *qsiz, const armpl_int_t *tlvls,
             const armpl_int_t *curlvl, const armpl_int_t *curpbm, float *d,
             float *q, const armpl_int_t *ldq, armpl_int_t *indxq,
             const float *rho, const armpl_int_t *cutpnt, float *qstore,
             armpl_int_t *qptr, const armpl_int_t *prmptr,
             const armpl_int_t *perm, const armpl_int_t *givptr,
             const armpl_int_t *givcol, const float *givnum, float *work,
             armpl_int_t *iwork, armpl_int_t *info);
```

Parameters

ICOMPQ Input parameter.

ICOMPQ is INTEGER

= 0: Compute eigenvalues only. = 1: Compute eigenvectors of original dense symmetric matrix also. On entry, Q contains the orthogonal matrix used to reduce the original matrix to tridiagonal form.

N Input parameter.

N is INTEGER

The dimension of the symmetric tridiagonal matrix. N >= 0.

QSIZ Input parameter.

QSIZ is INTEGER

The dimension of the orthogonal matrix used to reduce the full matrix to tridiagonal form. QSIZ >= N if ICOMPQ = 1.

TLVLS Input parameter.

TLVLS is INTEGER

The total number of merging levels in the overall divide and conquer tree.

CURLVL Input parameter.

CURLVL is INTEGER

The current level in the overall merge routine, 0 <= CURLVL <= TLVLS.

CURPBM Input parameter.

CURPBM is INTEGER

The current problem in the current level in the overall merge routine (counting from upper left to lower right).

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry, the eigenvalues of the rank-1-perturbed matrix. On exit, the eigenvalues of the repaired matrix.

Q Input and output parameter.

Q is REAL

Q is an array, dimension (LDQ, N). On entry, the eigenvectors of the rank-1-perturbed matrix. On exit, the eigenvectors of the repaired tridiagonal matrix.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, N).

INDXQ Output parameter.

INDXQ is INTEGER array, dimension (N)

The permutation which will reintegrate the subproblem just solved back into sorted order, i.e., D(INDXQ(I = 1, N)) will be in ascending order.

RHO Input parameter.

RHO is REAL

The subdiagonal element used to create the rank-1 modification.

CUTPNT Input parameter.

CUTPNT is INTEGER

Contains the location of the last eigenvalue in the leading sub-matrix. min(1, N) <= CUTPNT <= N.

QSTORE Input and output parameter.

QSTORE is REAL

QSTORE is an array, dimension (N**2+1). Stores eigenvectors of submatrices encountered during divide and conquer, packed together. QPTR points to beginning of the submatrices.

QPTR Input and output parameter.

QPTR is INTEGER array, dimension (N+2)

List of indices pointing to beginning of submatrices stored in QSTORE. The submatrices are numbered starting at the bottom left of the divide and conquer tree, from left to right and bottom to top.

PRMPTR Input parameter.

PRMPTR is INTEGER array, dimension (N lg N)

Contains a list of pointers which indicate where in PERM a level's permutation is stored. PRMPTR(i+1) - PRMPTR(i) indicates the size of the permutation and also the size of the full, non-deflated problem.

PERM Input parameter.

PERM is INTEGER array, dimension (N lg N)

Contains the permutations (from deflation and sorting) to be applied to each eigenblock.

GIVPTR Input parameter.

GIVPTR is INTEGER array, dimension (N lg N)

Contains a list of pointers which indicate where in GIVCOL a level's Givens rotations are stored. GIVPTR(i+1) - GIVPTR(i) indicates the number of Givens rotations.

GIVCOL Input parameter.

GIVCOL is INTEGER array, dimension (2, N lg N)

Each pair of numbers indicates a pair of columns to take place in a Givens rotation.

GIVNUM Input parameter.

GIVNUM is REAL

GIVNUM is an array, dimension (2, N lg N). Each number indicates the S value to be used in the corresponding Givens rotation.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (3*N+2*QSIZ*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (4*N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = 1, an eigenvalue did not converge

Related Information

For this routine in other precisions, please see *claed7*, *dlaed7* and *zlaed7*.

4.17.459 slaed8

slaed8 merges the two sets of eigenvalues together into a single sorted set. Then it tries to deflate the size of the problem. There are two ways in which deflation can occur: when two or more eigenvalues are close together or if there is a tiny element in the Z vector. For each such occurrence the order of the related secular equation problem is reduced by one.

Syntax

Fortran specification:

```
use armpl_library

subroutine slaed8(ICOMPQ, K, N, QSIZ, D, Q, LDQ, INDXQ, RHO, CUTPNT, Z,
                  DLAMDA, Q2, LDQ2, W, PERM, GIVPTR, GIVCOL, GIVNUM, INDXP,
                  INDX, INFO)
```

C specification:

```
#include "armpl.h"

void slaed8_(const armpl_int_t *icompq, armpl_int_t *k, const armpl_int_t *n,
             const armpl_int_t *qsiz, float *d, float *q,
             const armpl_int_t *ldq, const armpl_int_t *indxq, float *rho,
             const armpl_int_t *cutpnt, const float *z, float *dlamda,
             float *q2, const armpl_int_t *ldq2, float *w, armpl_int_t *perm,
             armpl_int_t *givptr, armpl_int_t *givcol, float *givnum,
             armpl_int_t *indxp, armpl_int_t *indx, armpl_int_t *info);
```

Parameters

ICOMPQ Input parameter.

ICOMPQ is INTEGER

= 0: Compute eigenvalues only. = 1: Compute eigenvectors of original dense symmetric matrix also. On entry, Q contains the orthogonal matrix used to reduce the original matrix to tridiagonal form.

K Output parameter.

K is INTEGER

The number of non-deflated eigenvalues, and the order of the related secular equation.

N Input parameter.

N is INTEGER

The dimension of the symmetric tridiagonal matrix. N >= 0.

QSIZ Input parameter.

QSIZ is INTEGER

The dimension of the orthogonal matrix used to reduce the full matrix to tridiagonal form. QSIZ >= N if ICOMPQ = 1.

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry, the eigenvalues of the two submatrices to be combined. On exit, the trailing (N-K) updated eigenvalues (those which were deflated) sorted into increasing order.

Q Input and output parameter.

Q is REAL

Q is an array, dimension (LDQ, N). If ICOMPQ = 0, Q is not referenced. Otherwise, on entry, Q contains the eigenvectors of the partially solved system which has been previously updated in matrix multiplies with other partially solved eigensystems. On exit, Q contains the trailing (N-K) updated eigenvectors (those which were deflated) in its last N-K columns.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, N).

INDXQ Input parameter.

INDXQ is INTEGER array, dimension (N)

The permutation which separately sorts the two sub-problems in D into ascending order. Note that elements in the second half of this permutation must first have CUTPNT added to their values in order to be accurate.

RHO Input and output parameter.

RHO is REAL

On entry, the off-diagonal element associated with the rank-1 cut which originally split the two submatrices which are now being recombined. On exit, RHO has been modified to the value required by SLAED3.

CUTPNT Input parameter.

CUTPNT is INTEGER

The location of the last eigenvalue in the leading sub-matrix. min(1, N) <= CUTPNT <= N.

Z Input parameter.

Z is REAL

Z is an array, dimension (N). On entry, Z contains the updating vector (the last row of the first sub-eigenvector matrix and the first row of the second sub-eigenvector matrix). On exit, the contents of Z are destroyed by the updating process.

DLAMDA Output parameter.

DLAMDA is REAL

DLAMDA is an array, dimension (N). A copy of the first K eigenvalues which will be used by SLAED3 to form the secular equation.

Q2 Output parameter.

Q2 is REAL

Q2 is an array, dimension (LDQ2, N). If ICOMPQ = 0, Q2 is not referenced. Otherwise, a copy of the first K eigenvectors which will be used by SLAED7 in a matrix multiply (SGEMM) to update the new eigenvectors.

LDQ2 Input parameter.

LDQ2 is INTEGER

The leading dimension of the array Q2. LDQ2 >= max(1, N).

W Output parameter.

W is REAL

W is an array, dimension (N). The first k values of the final deflation-altered z-vector and will be passed to SLAED3.

PERM Output parameter.

PERM is INTEGER array, dimension (N)

The permutations (from deflation and sorting) to be applied to each eigenblock.

GIVPTR Output parameter.

GIVPTR is INTEGER

The number of Givens rotations which took place in this subproblem.

GIVCOL Output parameter.

GIVCOL is INTEGER array, dimension (2, N)

Each pair of numbers indicates a pair of columns to take place in a Givens rotation.

GIVNUM Output parameter.

GIVNUM is REAL

GIVNUM is an array, dimension (2, N). Each number indicates the S value to be used in the corresponding Givens rotation.

INDXP Output parameter.

INDXP is INTEGER array, dimension (N)

The permutation used to place deflated values of D at the end of the array. INDXP(1:K) points to the nondeflated D-values and INDXP(K+1:N) points to the deflated eigenvalues.

INDX Output parameter.

INDX is INTEGER array, dimension (N)

The permutation used to sort the contents of D into ascending order.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *claed8*, *dlaed8* and *zlaed8*.

4.17.460 slaed9

slaed9 finds the roots of the secular equation, as defined by the values in D, Z, and RHO, between KSTART and KSTOP. It makes the appropriate calls to SLAED4 and then stores the new matrix of eigenvectors for use in calculating the next level of Z vectors.

Syntax

Fortran specification:

```
use armpl_library

subroutine slaed9(K, KSTART, KSTOP, N, D, Q, LDQ, RHO, DLAMDA, W, S, LDS,
                  INFO)
```

C specification:

```
#include "armpl.h"

void slaed9_(const armpl_int_t *k, const armpl_int_t *kstart,
             const armpl_int_t *kstop, const armpl_int_t *n, float *d,
             float *q, const armpl_int_t *ldq, const float *rho,
             const float *dlamda, const float *w, float *s,
             const armpl_int_t *lds, armpl_int_t *info);
```

Parameters

K Input parameter.

K is INTEGER

The number of terms in the rational function to be solved by SLAED4. K >= 0.

KSTART Input parameter.

KSTART is INTEGER

KSTOP Input parameter.

KSTOP is INTEGER

The updated eigenvalues Lambda(I), KSTART <= I <= KSTOP are to be computed. 1 <= KSTART <= KSTOP <= K.

N Input parameter.

N is INTEGER

The number of rows and columns in the Q matrix. N >= K (delation may result in N > K).

D Output parameter.

D is REAL

D is an array, dimension (N). D(I) contains the updated eigenvalues for KSTART <= I <= KSTOP.

Q Output parameter.

Q is REAL

Q is an array, dimension (LDQ, N) .

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, N).

RHO Input parameter.

RHO is REAL

The value of the parameter in the rank one update equation. RHO >= 0 required.

DLAMDA Input parameter.

DLAMDA is REAL

DLAMDA is an array, dimension (K). The first K elements of this array contain the old roots of the deflated updating problem. These are the poles of the secular equation.

W Input parameter.

W is REAL

W is an array, dimension (K). The first K elements of this array contain the components of the deflation-adjusted updating vector.

S Output parameter.

S is REAL

S is an array, dimension (LDS, K). Will contain the eigenvectors of the repaired matrix which will be stored for subsequent Z vector calculation and multiplied by the previously accumulated eigenvectors to update the system.

LDS Input parameter.

LDS is INTEGER

The leading dimension of S. LDS >= max(1, K).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = 1, an eigenvalue did not converge

Related Information

For this routine in other precisions, please see *dlaed9*.

4.17.461 slaeda

slaeda computes the Z vector corresponding to the merge step in the CURLVLth step of the merge process with TLVLS steps for the CURPBMth problem.

Syntax

Fortran specification:

```
use armpl_library

subroutine slaeda(N, TLVLS, CURLVL, CURPBM, PRMPTR, PERM, GIVPTR, GIVCOL,
                  GIVNUM, Q, QPTR, Z, ZTEMP, INFO)
```

C specification:

```
#include "armpl.h"

void slaeda_(const armpl_int_t *n, const armpl_int_t *tlvls,
             const armpl_int_t *curlvl, const armpl_int_t *curpbm,
             const armpl_int_t *prmptr, const armpl_int_t *perm,
             const armpl_int_t *givptr, const armpl_int_t *givcol,
             const float *givnum, const float *q, const armpl_int_t *qptr,
             float *z, float *ztemp, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The dimension of the symmetric tridiagonal matrix. N >= 0.

TLVLS Input parameter.

TLVLS is INTEGER

The total number of merging levels in the overall divide and conquer tree.

CURLVL Input parameter.

CURLVL is INTEGER

The current level in the overall merge routine, 0 <= curlvl <= tlvls.

CURPBM Input parameter.

CURPBM is INTEGER

The current problem in the current level in the overall merge routine (counting from upper left to lower right).

PRMPTR Input parameter.

PRMPTR is INTEGER array, dimension (N lg N)

Contains a list of pointers which indicate where in PERM a level's permutation is stored. PRMPTR(i+1) - PRMPTR(i) indicates the size of the permutation and incidentally the size of the full, non-deflated problem.

PERM Input parameter.

PERM is INTEGER array, dimension (N lg N)

Contains the permutations (from deflation and sorting) to be applied to each eigenblock.

GIVPTR Input parameter.

GIVPTR is INTEGER array, dimension (N lg N)

Contains a list of pointers which indicate where in GIVCOL a level's Givens rotations are stored. GIVPTR(i+1) - GIVPTR(i) indicates the number of Givens rotations.

GIVCOL Input parameter.

GIVCOL is INTEGER array, dimension (2, N lg N)

Each pair of numbers indicates a pair of columns to take place in a Givens rotation.

GIVNUM Input parameter.

GIVNUM is REAL

GIVNUM is an array, dimension (2, N lg N). Each number indicates the S value to be used in the corresponding Givens rotation.

Q Input parameter.

Q is REAL

Q is an array, dimension (N**2). Contains the square eigenblocks from previous levels, the starting positions for blocks are given by QPTR.

QPTR Input parameter.

QPTR is INTEGER array, dimension (N+2)

Contains a list of pointers which indicate where in Q an eigenblock is stored. SQRT(QPTR(i+1) - QPTR(i)) indicates the size of the block.

Z Output parameter.

Z is REAL

Z is an array, dimension (N). On output this vector contains the updating vector (the last row of the first sub-eigenvector matrix and the first row of the second sub-eigenvector matrix).

ZTEMP Output parameter.

ZTEMP is REAL

ZTEMP is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *dlaeda*.

4.17.462 slaein

`slaein` uses inverse iteration to find a right or left eigenvector corresponding to the eigenvalue (WR,WI) of a real upper Hessenberg matrix H.

Syntax

Fortran specification:

```
use armpl_library

subroutine slaein(RIGHTV, NOINIT, N, H, LDH, WR, WI, VR, VI, B, LDB, WORK,
                  EPS3, SMLNUM, BIGNUM, INFO)
```

C specification:

```
#include "armpl.h"

void slaein_(const armpl_int_t *rightv, const armpl_int_t *noinit,
             const armpl_int_t *n, const float *h, const armpl_int_t *ldh,
             const float *wr, const float *wi, float *vr, float *vi, float *b,
             const armpl_int_t *ldb, float *work, const float *eps3,
             const float *smlnum, const float *bignum, armpl_int_t *info);
```

Parameters

RIGHTV Input parameter.

RIGHTV is LOGICAL

= .TRUE. : compute right eigenvector; = .FALSE.: compute left eigenvector.

NOINIT Input parameter.

NOINIT is LOGICAL

= .TRUE. : no initial vector supplied in (VR, VI). = .FALSE.: initial vector supplied in (VR, VI).

N Input parameter.

N is INTEGER

The order of the matrix H. N >= 0.

H Input parameter.

H is REAL

H is an array, dimension (LDH, N). The upper Hessenberg matrix H.

LDH Input parameter.

LDH is INTEGER

The leading dimension of the array H. LDH >= max(1, N).

WR Input parameter.

WR is REAL

WI Input parameter.

WI is REAL

The real and imaginary parts of the eigenvalue of H whose corresponding right or left eigenvector is to be computed.

VR Input and output parameter.

VR is REAL

VR is an array, dimension (N) .

VI Input and output parameter.

VI is REAL

VI is an array, dimension (N). On entry, if NOINIT = .FALSE. and WI = 0.0, VR must contain a real starting vector for inverse iteration using the real eigenvalue WR; if NOINIT = .FALSE. and WI.ne.0.0, VR and VI must contain the real and imaginary parts of a complex starting vector for inverse iteration using the complex eigenvalue (WR, WI); otherwise VR and VI need not be set. On exit, if WI = 0.0 (real eigenvalue), VR contains the computed real eigenvector; if WI.ne.0.0 (complex eigenvalue), VR and VI contain the real and imaginary parts of the computed complex eigenvector. The eigenvector is normalized so that the component of largest magnitude has magnitude 1; here the magnitude of a complex number (x,y) is taken to be |x| + |y|. VI is not referenced if WI = 0.0.

B Output parameter.

B is REAL

B is an array, dimension (LDB, N) .

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= N+1.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (N) .

EPS3 Input parameter.

EPS3 is REAL

A small machine-dependent value which is used to perturb close eigenvalues, and to replace zero pivots.

SMLNUM Input parameter.

SMLNUM is REAL

A machine-dependent value close to the underflow threshold.

BIGNUM Input parameter.

BIGNUM is REAL

A machine-dependent value close to the overflow threshold.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit = 1: inverse iteration did not converge; VR is set to the last iterate, and so is VI if WI.ne.0.0.

Related Information

For this routine in other precisions, please see *claein*, *dlaein* and *zlaein*.

4.17.463 slaev2

slaev2 computes the eigendecomposition of a 2-by-2 symmetric matrix

```
[  A   B  ]
[  B   C  ].
```

On return, RT1 is the eigenvalue of larger absolute value, RT2 is the eigenvalue of smaller absolute value, and (CS1,SN1) is the unit right eigenvector for RT1, giving the decomposition

```
[ CS1  SN1 ] [  A   B  ] [ CS1 -SN1 ]  =  [ RT1  0  ]
[-SN1  CS1 ] [  B   C  ] [ SN1  CS1 ]     [  0  RT2 ].
```

Syntax

Fortran specification:

```
use armpl_library

subroutine slaev2(A, B, C, RT1, RT2, CS1, SN1)
```

C specification:

```
#include "armpl.h"

void slaev2_(const float *a, const float *b, const float *c, float *rt1,
             float *rt2, float *cs1, float *sn1);
```

Parameters

A Input parameter.

A is REAL

The (1,1) element of the 2-by-2 matrix.

B Input parameter.

B is REAL

The (1,2) element and the conjugate of the (2,1) element of the 2-by-2 matrix.

C Input parameter.

C is REAL

The (2,2) element of the 2-by-2 matrix.

RT1 Output parameter.

RT1 is REAL

The eigenvalue of larger absolute value.

RT2 Output parameter.

RT2 is REAL

The eigenvalue of smaller absolute value.

CS1 Output parameter.

CS1 is REAL

SN1 Output parameter.

SN1 is REAL

The vector (CS1, SN1) is a unit right eigenvector for RT1.

Related Information

For this routine in other precisions, please see *claev2*, *dlaev2* and *zlaev2*.

4.17.464 slaexc

slaexc swaps adjacent diagonal blocks T11 and T22 of order 1 or 2 in an upper quasi-triangular matrix T by an orthogonal similarity transformation.

T must be in Schur canonical form, that is, block upper triangular with 1-by-1 and 2-by-2 diagonal blocks; each 2-by-2 diagonal block has its diagonal elemnts equal and its off-diagonal elements of opposite sign.

Syntax

Fortran specification:

```
use armpl_library

subroutine slaexc(WANTQ, N, T, LDT, Q, LDQ, J1, N1, N2, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void slaexc_(const armpl_int_t *wantq, const armpl_int_t *n, float *t,
             const armpl_int_t *ldt, float *q, const armpl_int_t *ldq,
             const armpl_int_t *j1, const armpl_int_t *n1,
             const armpl_int_t *n2, float *work, armpl_int_t *info);
```

Parameters

WANTQ Input parameter.

WANTQ is LOGICAL

= .TRUE. : accumulate the transformation in the matrix Q; = .FALSE.: do not accumulate the transformation.

N Input parameter.

N is INTEGER

The order of the matrix T. N >= 0.

T Input and output parameter.

T is REAL

T is an array, dimension (LDT, N). On entry, the upper quasi-triangular matrix T, in Schur canonical form. On exit, the updated matrix T, again in Schur canonical form.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

Q Input and output parameter.

Q is REAL

Q is an array, dimension (LDQ, N). On entry, if WANTQ is .TRUE., the orthogonal matrix Q. On exit, if WANTQ is .TRUE., the updated matrix Q. If WANTQ is .FALSE., Q is not referenced.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= 1; and if WANTQ is .TRUE., LDQ >= N.

J1 Input parameter.

J1 is INTEGER

The index of the first row of the first block T11.

N1 Input parameter.

N1 is INTEGER

The order of the first block T11. N1 = 0, 1 or 2.

N2 Input parameter.

N2 is INTEGER

The order of the second block T22. N2 = 0, 1 or 2.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit = 1: the transformed matrix T would be too far from Schur form; the blocks are not swapped and T and Q are unchanged.

Related Information

For this routine in other precisions, please see *dlaexc*.

4.17.465 slag2

slag2 computes the eigenvalues of a 2 x 2 generalized eigenvalue problem A - w B, with scaling as necessary to avoid over-/underflow.

The scaling factor "s" results in a modified eigenvalue equation

```
s A - w B
```

where s is a non-negative scaling factor chosen so that w, w B, and s A do not overflow and, if possible, do not underflow, either.

Syntax

Fortran specification:

```
use armpl_library

subroutine slag2(A, LDA, B, LDB, SAFMIN, SCALE1, SCALE2, WR1, WR2, WI)
```

C specification:

```
#include "armpl.h"

void slag2_(const float *a, const armpl_int_t *lda, const float *b,
            const armpl_int_t *ldb, const float *safmin, float *scale1,
            float *scale2, float *wr1, float *wr2, float *wi);
```

Parameters

A Input parameter.

A is REAL

A is an array, dimension (LDA, 2). On entry, the 2 x 2 matrix A. It is assumed that its 1-norm is less than 1/SAFMIN. Entries less than sqrt(SAFMIN)*norm(A) are subject to being treated as zero.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= 2.

B Input parameter.

B is REAL

B is an array, dimension (LDB, 2). On entry, the 2 x 2 upper triangular matrix B. It is assumed that the one-norm of B is less than 1/SAFMIN. The diagonals should be at least sqrt(SAFMIN) times the largest element of B (in absolute value); if a diagonal is smaller than that, then +/- sqrt(SAFMIN) will be used instead of that diagonal.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= 2.

SAFMIN Input parameter.

SAFMIN is REAL

The smallest positive number s.t. 1/SAFMIN does not overflow. (This should always be SLAMCH('S') – it is an argument in order to avoid having to call SLAMCH frequently.)

SCALE1 Output parameter.

SCALE1 is REAL

A scaling factor used to avoid over-/underflow in the eigenvalue equation which defines the first eigenvalue. If the eigenvalues are complex, then the eigenvalues are (WR1 +/- WI i) / SCALE1 (which may lie outside the exponent range of the machine), SCALE1=SCALE2, and SCALE1 will always be positive. If the eigenvalues are real, then the first (real) eigenvalue is WR1 / SCALE1 , but this may overflow or underflow, and in fact, SCALE1 may be zero or less than the underflow threshold if the exact eigenvalue is sufficiently large.

SCALE2 Output parameter.

SCALE2 is REAL

A scaling factor used to avoid over-/underflow in the eigenvalue equation which defines the second eigenvalue. If the eigenvalues are complex, then SCALE2=SCALE1. If the eigenvalues are real, then the second (real) eigenvalue is WR2 / SCALE2 , but this may overflow or underflow, and in fact, SCALE2 may be zero or less than the underflow threshold if the exact eigenvalue is sufficiently large.

WR1 Output parameter.

WR1 is REAL

If the eigenvalue is real, then WR1 is SCALE1 times the eigenvalue closest to the (2,2) element of A B**(-1). If the eigenvalue is complex, then WR1=WR2 is SCALE1 times the real part of the eigenvalues.

WR2 Output parameter.

WR2 is REAL

If the eigenvalue is real, then WR2 is SCALE2 times the other eigenvalue. If the eigenvalue is complex, then WR1=WR2 is SCALE1 times the real part of the eigenvalues.

WI Output parameter.

WI is REAL

If the eigenvalue is real, then WI is zero. If the eigenvalue is complex, then WI is SCALE1 times the imaginary part of the eigenvalues. WI will always be non-negative.

Related Information

For this routine in other precisions, please see *dlag2*.

4.17.466 slag2d

slag2d converts a SINGLE PRECISION matrix, SA, to a DOUBLE PRECISION matrix, A.

Note that while it is possible to overflow while converting from double to single, it is not possible to overflow when converting from single to double.

This is an auxiliary routine so there is no argument checking.

Syntax

Fortran specification:

```
use armpl_library

subroutine slag2d(M, N, SA, LDSA, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void slag2d_(const armpl_int_t *m, const armpl_int_t *n, const float *sa,
             const armpl_int_t *ldsa, double *a, const armpl_int_t *lda,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of lines of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

SA Input parameter.

SA is REAL

SA is an array, dimension (LDSA, N). On entry, the M-by-N coefficient matrix SA.

LDSA Input parameter.

LDSA is INTEGER

The leading dimension of the array SA. LDSA >= max(1, M).

A Output parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). On exit, the M-by-N coefficient matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit

Related Information

It also exists with a native C interface as *LAPACKE_slag2d*.

4.17.467 slags2

slags2 computes 2-by-2 orthogonal matrices U, V and Q, such that if (UPPER) then

```
U**T *A*Q = U**T *( A1 A2 )*Q = ( x  0  )
                  ( 0  A3 )     ( x  x  )
```

and

```
V**T*B*Q = V**T *( B1 B2 )*Q = ( x  0  )
                 ( 0  B3 )     ( x  x  )
```

or if (.NOT.UPPER) then

```
U**T *A*Q = U**T *( A1 0  )*Q = ( x  x  )
                  ( A2 A3 )     ( 0  x  )
```

and

```
V**T*B*Q = V**T*( B1 0  )*Q = ( x  x  )
                ( B2 B3 )     ( 0  x  )
```

The rows of the transformed A and B are parallel, where

```
U = (  CSU  SNU ), V = (  CSV SNV ), Q = (  CSQ   SNQ )
   ( -SNU  CSU )       ( -SNV CSV )       ( -SNQ   CSQ )
```

Z^T denotes the transpose of Z.

Syntax

Fortran specification:

```
use armpl_library

subroutine slags2(UPPER, A1, A2, A3, B1, B2, B3, CSU, SNU, CSV, SNV, CSQ,
                  SNQ)
```

C specification:

```
#include "armpl.h"

void slags2_(const armpl_int_t *upper, const float *a1, const float *a2,
             const float *a3, const float *b1, const float *b2,
             const float *b3, float *csu, float *snu, float *csv, float *snv,
             float *csq, float *snq);
```

Parameters

UPPER Input parameter.

UPPER is LOGICAL

= .TRUE.: the input matrices A and B are upper triangular. = .FALSE.: the input matrices A and B are lower triangular.

A1 Input parameter.

A1 is REAL

A2 Input parameter.

A2 is REAL

A3 Input parameter.

A3 is REAL

On entry, A1, A2 and A3 are elements of the input 2-by-2 upper (lower) triangular matrix A.

B1 Input parameter.

B1 is REAL

B2 Input parameter.

B2 is REAL

B3 Input parameter.

B3 is REAL

On entry, B1, B2 and B3 are elements of the input 2-by-2 upper (lower) triangular matrix B.

CSU Output parameter.

CSU is REAL

SNU Output parameter.

SNU is REAL

The desired orthogonal matrix U.

CSV Output parameter.

CSV is REAL

SNV Output parameter.

SNV is REAL

The desired orthogonal matrix V.

CSQ Output parameter.

CSQ is REAL

SNQ Output parameter.

SNQ is REAL

The desired orthogonal matrix Q.

Related Information

For this routine in other precisions, please see *clags2*, *dlags2* and *zlags2*.

4.17.468 slagtf

slagtf factorizes the matrix (T - lambda*I), where T is an n by n tridiagonal matrix and lambda is a scalar, as

```
T - lambda*I = PLU,
```

where P is a permutation matrix, L is a unit lower tridiagonal matrix with at most one non-zero sub-diagonal elements per column and U is an upper triangular matrix with at most two non-zero super-diagonal elements per column.

The factorization is obtained by Gaussian elimination with partial pivoting and implicit row scaling.

The parameter LAMBDA is included in the routine so that slagtf may be used, in conjunction with SLAGTS, to obtain eigenvectors of T by inverse iteration.

Syntax

Fortran specification:

```
use armpl_library

subroutine slagtf(N, A, LAMBDA, B, C, TOL, D, IN, INFO)
```

C specification:

```
#include "armpl.h"

void slagtf_(const armpl_int_t *n, float *a, const float *lambda, float *b,
             float *c, const float *tol, float *d, armpl_int_t *in,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix T.

A Input and output parameter.

A is REAL

A is an array, dimension (N). On entry, A must contain the diagonal elements of T.

On exit, A is overwritten by the n diagonal elements of the upper triangular matrix U of the factorization of T.

LAMBDA Input parameter.

LAMBDA is REAL

On entry, the scalar lambda.

B Input and output parameter.

B is REAL

B is an array, dimension (N-1). On entry, B must contain the (n-1) super-diagonal elements of T.

On exit, B is overwritten by the (n-1) super-diagonal elements of the matrix U of the factorization of T.

C Input and output parameter.

C is REAL

C is an array, dimension (N-1). On entry, C must contain the (n-1) sub-diagonal elements of T.

On exit, C is overwritten by the (n-1) sub-diagonal elements of the matrix L of the factorization of T.

TOL Input parameter.

TOL is REAL

On entry, a relative tolerance used to indicate whether or not the matrix (T - lambda*I) is nearly singular. TOL should normally be chose as approximately the largest relative error in the elements of T. For example, if the elements of T are correct to about 4 significant figures, then TOL should be set to about 5*10**(-4). If TOL is supplied as less than eps, where eps is the relative machine precision, then the value eps is used in place of TOL.

D Output parameter.

D is REAL

D is an array, dimension (N-2). On exit, D is overwritten by the (n-2) second super-diagonal elements of the matrix U of the factorization of T.

IN Output parameter.

IN is INTEGER array, dimension (N)

On exit, IN contains details of the permutation matrix P. If an interchange occurred at the kth step of the elimination, then IN(k) = 1, otherwise IN(k) = 0. The element IN(n) returns the smallest positive integer j such that

abs(u(j,j)).le. norm((T - lambda*I)(j))*TOL,

where norm(A(j)) denotes the sum of the absolute values of the jth row of the matrix A. If no such j exists then IN(n) is returned as zero. If IN(n) is returned as positive, then a diagonal element of U is small, indicating that (T - lambda*I) is singular or nearly singular,

INFO Output parameter.

INFO is INTEGER

= 0 : successful exit .lt. 0: if INFO = -k, the kth argument had an illegal value

Related Information

For this routine in other precisions, please see *dlagtf*.

4.17.469 slagtm

slagtm performs a matrix-vector product of the form

```
B := alpha * A * X + beta * B
```

where A is a tridiagonal matrix of order N, B and X are N by NRHS matrices, and alpha and beta are real scalars, each of which may be 0., 1., or -1.

Syntax

Fortran specification:

```
use armpl_library

subroutine slagtm(TRANS, N, NRHS, ALPHA, DL, D, DU, X, LDX, BETA, B, LDB)
```

C specification:

```
#include "armpl.h"

void slagtm_(const char *trans, const armpl_int_t *n, const armpl_int_t *nrhs,
             const float *alpha, const float *dl, const float *d,
             const float *du, const float *x, const armpl_int_t *ldx,
             const float *beta, float *b, const armpl_int_t *ldb, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the operation applied to A. = 'N': No transpose, B := alpha * A * X + beta * B = 'T': Transpose, B := alpha * A'* X + beta * B = 'C': Conjugate transpose = Transpose

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices X and B.

ALPHA Input parameter.

ALPHA is REAL

The scalar alpha. ALPHA must be 0., 1., or -1.; otherwise, it is assumed to be 0.

DL Input parameter.

DL is REAL

DL is an array, dimension (N-1). The (n-1) sub-diagonal elements of T.

D Input parameter.

D is REAL

D is an array, dimension (N). The diagonal elements of T.

DU Input parameter.

DU is REAL

DU is an array, dimension (N-1). The (n-1) super-diagonal elements of T.

X Input parameter.

X is REAL

X is an array, dimension (LDX, NRHS). The N by NRHS matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(N,1).

BETA Input parameter.

BETA is REAL

The scalar beta. BETA must be 0., 1., or -1.; otherwise, it is assumed to be 1.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the N by NRHS matrix B. On exit, B is overwritten by the matrix expression B := alpha * A * X + beta * B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(N,1).

Related Information

For this routine in other precisions, please see *clagtm*, *dlagtm* and *zlagtm*.

4.17.470 slagts

slagts may be used to solve one of the systems of equations

```
(T - lambda*I)*x = y   or   (T - lambda*I)**T*x = y,
```

where T is an n by n tridiagonal matrix, for x, following the factorization of (T - lambda*I) as

```
(T - lambda*I) = P*L*U ,
```

by routine SLAGTF. The choice of equation to be solved is controlled by the argument JOB, and in each case there is an option to perturb zero or very small diagonal elements of U, this option being intended for use in applications such as inverse iteration.

Syntax

Fortran specification:

```
use armpl_library

subroutine slagts(JOB, N, A, B, C, D, IN, Y, TOL, INFO)
```

C specification:

```
#include "armpl.h"

void slagts_(const armpl_int_t *job, const armpl_int_t *n, const float *a,
             const float *b, const float *c, const float *d,
             const armpl_int_t *in, float *y, float *tol, armpl_int_t *info);
```

Parameters

JOB Input parameter.

JOB is INTEGER

Specifies the job to be performed by SLAGTS as follows: = 1: The equations (T - lambda*I)x = y are to be solved, but diagonal elements of U are not to be perturbed. = -1: The equations (T - lambda*I)x = y are to be solved and, if overflow would otherwise occur, the diagonal elements of U are to be perturbed. See argument TOL below. = 2: The equations $(T - lambda*I)^T$ x = y are to be solved, but diagonal elements of U are not to be perturbed. = -2: The equations $(T - lambda*I)^T$ x = y are to be solved and, if overflow would otherwise occur, the diagonal elements of U are to be perturbed. See argument TOL below.

N Input parameter.

N is INTEGER

The order of the matrix T.

A Input parameter.

A is REAL

A is an array, dimension (N). On entry, A must contain the diagonal elements of U as returned from SLAGTF.

B Input parameter.

B is REAL

B is an array, dimension (N-1). On entry, B must contain the first super-diagonal elements of U as returned from SLAGTF.

C Input parameter.

C is REAL

C is an array, dimension (N-1). On entry, C must contain the sub-diagonal elements of L as returned from SLAGTF.

D Input parameter.

D is REAL

D is an array, dimension (N-2). On entry, D must contain the second super-diagonal elements of U as returned from SLAGTF.

IN Input parameter.

IN is INTEGER array, dimension (N)

On entry, IN must contain details of the matrix P as returned from SLAGTF.

Y Input and output parameter.

Y is REAL

Y is an array, dimension (N). On entry, the right hand side vector y. On exit, Y is overwritten by the solution vector x.

TOL Input and output parameter.

TOL is REAL

On entry, with JOB .lt. 0, TOL should be the minimum perturbation to be made to very small diagonal elements of U. TOL should normally be chosen as about eps*norm(U), where eps is the relative machine precision, but if TOL is supplied as non-positive, then it is reset to eps*max(abs(u(i,j))). If JOB .gt. 0 then TOL is not referenced.

On exit, TOL is changed as described above, only if TOL is non-positive on entry. Otherwise TOL is unchanged.

INFO Output parameter.

INFO is INTEGER

= 0 : successful exit .lt. 0: if INFO = -i, the i-th argument had an illegal value .gt. 0: overflow would occur when computing the INFO(th) element of the solution vector x. This can only occur when JOB is supplied as positive and either means that a diagonal element of U is very small, or that the elements of the right-hand side vector y are very large.

Related Information

For this routine in other precisions, please see *dlagts*.

4.17.471 slagv2

slagv2 computes the Generalized Schur factorization of a real 2-by-2 matrix pencil (A,B) where B is upper triangular. This routine computes orthogonal (rotation) matrices given by CSL, SNL and CSR, SNR such that

1) if the pencil (A,B) has two real eigenvalues (include 0/0 or 1/0

```
 types), then

[ a11 a12 ] := [  CSL  SNL ] [ a11 a12 ] [  CSR -SNR ]
[  0  a22 ]    [ -SNL  CSL ] [ a21 a22 ] [  SNR  CSR ]

[ b11 b12 ] := [  CSL  SNL ] [ b11 b12 ] [  CSR -SNR ]
[  0  b22 ]    [ -SNL  CSL ] [  0  b22 ] [  SNR  CSR ],
```

2) if the pencil (A,B) has a pair of complex conjugate eigenvalues,

```
 then

[ a11 a12 ] := [  CSL  SNL ] [ a11 a12 ] [  CSR -SNR ]
[ a21 a22 ]    [ -SNL  CSL ] [ a21 a22 ] [  SNR  CSR ]

[ b11  0  ] := [  CSL  SNL ] [ b11 b12 ] [  CSR -SNR ]
[  0  b22 ]    [ -SNL  CSL ] [  0  b22 ] [  SNR  CSR ]

where b11 >= b22 > 0.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine slagv2(A, LDA, B, LDB, ALPHAR, ALPHAI, BETA, CSL, SNL, CSR, SNR)
```

C specification:

```
#include "armpl.h"

void slagv2_(float *a, const armpl_int_t *lda, float *b,
             const armpl_int_t *ldb, float *alphar, float *alphai,
             float *beta, float *csl, float *snl, float *csr, float *snr);
```

Parameters

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, 2). On entry, the 2 x 2 matrix A. On exit, A is overwritten by the ''A-part" of the generalized Schur form.

LDA Input parameter.

LDA is INTEGER

THe leading dimension of the array A. LDA >= 2.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, 2). On entry, the upper triangular 2 x 2 matrix B. On exit, B is overwritten by the ''B-part" of the generalized Schur form.

LDB Input parameter.

LDB is INTEGER

THe leading dimension of the array B. LDB >= 2.

ALPHAR Output parameter.

ALPHAR is REAL

ALPHAR is an array, dimension (2) .

ALPHAI Output parameter.

ALPHAI is REAL

ALPHAI is an array, dimension (2) .

BETA Output parameter.

BETA is REAL

BETA is an array, dimension (2). (ALPHAR(k)+i*ALPHAI(k))/BETA(k) are the eigenvalues of the pencil (A, B), k=1,2, i = sqrt(-1). Note that BETA(k) may be zero.

CSL Output parameter.

CSL is REAL

The cosine of the left rotation matrix.

SNL Output parameter.

SNL is REAL

The sine of the left rotation matrix.

CSR Output parameter.

CSR is REAL

The cosine of the right rotation matrix.

SNR Output parameter.

SNR is REAL

The sine of the right rotation matrix.

Related Information

For this routine in other precisions, please see *dlagv2*.

4.17.472 slahqr

```
SLAHQR is an auxiliary routine called by SHSEQR to update the
   eigenvalues and Schur decomposition already computed by SHSEQR, by
   dealing with the Hessenberg submatrix in rows and columns ILO to
   IHI.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine slahqr(WANTT, WANTZ, N, ILO, IHI, H, LDH, WR, WI, ILOZ, IHIZ, Z,
                  LDZ, INFO)
```

C specification:

```
#include "armpl.h"

void slahqr_(const armpl_int_t *wantt, const armpl_int_t *wantz,
             const armpl_int_t *n, const armpl_int_t *ilo,
             const armpl_int_t *ihi, float *h, const armpl_int_t *ldh,
             float *wr, float *wi, const armpl_int_t *iloz,
             const armpl_int_t *ihiz, float *z, const armpl_int_t *ldz,
             armpl_int_t *info);
```

Parameters

WANTT Input parameter.

WANTT is LOGICAL

= .TRUE. : the full Schur form T is required; = .FALSE.: only eigenvalues are required.

WANTZ Input parameter.

WANTZ is LOGICAL

= .TRUE. : the matrix of Schur vectors Z is required; = .FALSE.: Schur vectors are not required.

N Input parameter.

N is INTEGER

The order of the matrix H. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

It is assumed that H is already upper quasi-triangular in rows and columns IHI+1:N, and that H(ILO,ILO-1) = 0 (unless ILO = 1). SLAHQR works primarily with the Hessenberg submatrix in rows and columns ILO to IHI, but applies transformations to all of H if WANTT is .TRUE.. 1 <= ILO <= max(1, IHI); IHI <= N.

H Input and output parameter.

H is REAL

H is an array, dimension (LDH, N). On entry, the upper Hessenberg matrix H. On exit, if INFO is zero and if WANTT is .TRUE., H is upper quasi-triangular in rows and columns ILO:IHI, with any 2-by-2 diagonal blocks in standard form. If INFO is zero and WANTT is .FALSE., the contents of H are unspecified on exit. The output state of H if INFO is nonzero is given below under the description of INFO.

LDH Input parameter.

LDH is INTEGER

The leading dimension of the array H. LDH >= max(1, N).

WR Output parameter.

WR is REAL

WR is an array, dimension (N) .

WI Output parameter.

WI is REAL

WI is an array, dimension (N). The real and imaginary parts, respectively, of the computed eigenvalues ILO to IHI are stored in the corresponding elements of WR and WI. If two eigenvalues are computed as a complex conjugate pair, they are stored in consecutive elements of WR and WI, say the i-th and (i+1)th, with WI(i) > 0 and WI(i+1) < 0. If WANTT is .TRUE., the eigenvalues are stored in the same order as on the diagonal of the Schur form returned in H, with WR(i) = H(i,i), and, if H(i:i+1,i:i+1) is a 2-by-2 diagonal block, WI(i) = sqrt(H(i+1,i)*H(i,i+1)) and WI(i+1) = -WI(i).

ILOZ Input parameter.

ILOZ is INTEGER

IHIZ Input parameter.

IHIZ is INTEGER

Specify the rows of Z to which transformations must be applied if WANTZ is .TRUE.. 1 <= ILOZ <= ILO; IHI <= IHIZ <= N.

Z Input and output parameter.

Z is REAL

Z is an array, dimension (LDZ, N). If WANTZ is .TRUE., on entry Z must contain the current matrix Z of transformations accumulated by SHSEQR, and on exit Z has been updated; transformations are applied only to the submatrix Z(ILOZ:IHIZ,ILO:IHI). If WANTZ is .FALSE., Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit .GT. 0: If INFO = i, SLAHQR failed to compute all the eigenvalues ILO to IHI in a total of 30 iterations per eigenvalue; elements i+1:ihi of WR and WI contain those eigenvalues which have been successfully computed.

If INFO .GT. 0 and WANTT is .FALSE., then on exit, the remaining unconverged eigenvalues are the eigenvalues of the upper Hessenberg matrix rows and columns ILO thorugh INFO of the final, output value of H.

If INFO .GT. 0 and WANTT is .TRUE., then on exit (*) (initial value of H)*U = U*(final value of H) where U is an orthognal matrix. The final value of H is upper Hessenberg and triangular in rows and columns INFO+1 through IHI.

If INFO .GT. 0 and WANTZ is .TRUE., then on exit (final value of Z) = (initial value of Z)*U where U is the orthogonal matrix in (*) (regardless of the value of WANTT.)

Related Information

For this routine in other precisions, please see *clahqr*, *dlahqr* and *zlahqr*.

4.17.473 slahr2

slahr2 reduces the first NB columns of A real general n-BY-(n-k+1) matrix A so that elements below the k-th subdiagonal are zero. The reduction is performed by an orthogonal similarity transformation Q^T * A * Q. The routine returns the matrices V and T which determine Q as a block reflector I - V*T*V^T , and also the matrix Y = A * V * T.

This is an auxiliary routine called by SGEHRD.

Syntax

Fortran specification:

```
use armpl_library

subroutine slahr2(N, K, NB, A, LDA, TAU, T, LDT, Y, LDY)
```

C specification:

```
#include "armpl.h"

void slahr2_(const armpl_int_t *n, const armpl_int_t *k,
             const armpl_int_t *nb, float *a, const armpl_int_t *lda,
             float *tau, float *t, const armpl_int_t *ldt, float *y,
             const armpl_int_t *ldy);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A.

K Input parameter.

K is INTEGER

The offset for the reduction. Elements below the k-th subdiagonal in the first NB columns are reduced to zero. K < N.

NB Input parameter.

NB is INTEGER

The number of columns to be reduced.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA,N-K+1). On entry, the n-by-(n-k+1) general matrix A. On exit, the elements on and above the k-th subdiagonal in the first NB columns are overwritten with the corresponding elements of the reduced matrix; the elements below the k-th subdiagonal, with the array TAU, represent the matrix Q as a product of elementary reflectors. The other columns of A are unchanged. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TAU Output parameter.

TAU is REAL

TAU is an array, dimension (NB). The scalar factors of the elementary reflectors. See Further Details.

T Output parameter.

T is REAL

T is an array, dimension (LDT, NB). The upper triangular matrix T.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= NB.

Y Output parameter.

Y is REAL

Y is an array, dimension (LDY, NB). The n-by-nb matrix Y.

LDY Input parameter.

LDY is INTEGER

The leading dimension of the array Y. LDY >= N.

Related Information

For this routine in other precisions, please see *clahr2*, *dlahr2* and *zlahr2*.

4.17.474 slaic1

slaic1 applies one step of incremental condition estimation in its simplest version:

Let x, twonorm(x) = 1, be an approximate singular vector of an j-by-j lower triangular matrix L, such that

```
twonorm(L*x) = sest
```

Then slaic1 computes sestpr, s, c such that the vector

```
       [ s*x ]
xhat = [  c  ]
```

is an approximate singular vector of

```
       [ L       0    ]
Lhat = [ w**T gamma ]
```

in the sense that

```
twonorm(Lhat*xhat) = sestpr.
```

Depending on JOB, an estimate for the largest or smallest singular value is computed.

Note that [s c]^T and sestpr**2 is an eigenpair of the system

```
diag(sest*sest, 0) + [alpha  gamma] * [ alpha ]
                                      [ gamma ]
```

where alpha = x^T *w.

Syntax

Fortran specification:

```
use armpl_library

subroutine slaic1(JOB, J, X, SEST, W, GAMMA, SESTPR, S, C)
```

C specification:

```
#include "armpl.h"

void slaic1_(const armpl_int_t *job, const armpl_int_t *j, const float *x,
             const float *sest, const float *w, const float *gamma,
             float *sestpr, float *s, float *c);
```

Parameters

JOB Input parameter.

JOB is INTEGER

= 1: an estimate for the largest singular value is computed. = 2: an estimate for the smallest singular value is computed.

J Input parameter.

J is INTEGER

Length of X and W

X Input parameter.

X is REAL

X is an array, dimension (J). The j-vector x.

SEST Input parameter.

SEST is REAL

Estimated singular value of j by j matrix L

W Input parameter.

W is REAL

W is an array, dimension (J). The j-vector w.

GAMMA Input parameter.

GAMMA is REAL

The diagonal element gamma.

SESTPR Output parameter.

SESTPR is REAL

Estimated singular value of (j+1) by (j+1) matrix Lhat.

S Output parameter.

S is REAL

Sine needed in forming xhat.

C Output parameter.

C is REAL

Cosine needed in forming xhat.

Related Information

For this routine in other precisions, please see *claic1*, *dlaic1* and *zlaic1*.

4.17.475 slaisnan

This routine is not for general use. It exists solely to avoid over-optimization in SISNAN.

SLAISNAN checks for NaNs by comparing its two arguments for inequality. NaN is the only floating-point value where NaN != NaN returns .TRUE. To check for NaNs, pass the same variable as both arguments.

A compiler must assume that the two arguments are not the same variable, and the test will not be optimized away. Interprocedural or whole-program optimization may delete this test. The ISNAN functions will be replaced by the correct Fortran 03 intrinsic once the intrinsic is widely available.

Syntax

Fortran specification:

```
use armpl_library

logical function slaisnan(SIN1, SIN2)
```

C specification:

```
#include "armpl.h"

armpl_int_t slaisnan_(const float *sin1, const float *sin2);
```

Parameters

SIN1 Input parameter.

SIN1 is REAL

SIN2 Input parameter.

SIN2 is REAL

Two numbers to compare for inequality.

Related Information

For this routine in other precisions, please see *dlaisnan*.

4.17.476 slaln2

`slaln2` solves a system of the form (ca A - w D) X = s B or (ca A^T - w D) X = s B with possible scaling ("s") and perturbation of A. (A^T means A-transpose.)

A is an NA x NA real matrix, ca is a real scalar, D is an NA x NA real diagonal matrix, w is a real or complex value, and X and B are NA x 1 matrices – real if w is real, complex if w is complex. NA may be 1 or 2.

If w is complex, X and B are represented as NA x 2 matrices, the first column of each being the real part and the second being the imaginary part.

"s" is a scaling factor (.LE. 1), computed by `slaln2`, which is so chosen that X can be computed without overflow. X is further scaled if necessary to assure that norm(ca A - w D)*norm(X) is less than overflow.

If both singular values of (ca A - w D) are less than SMIN, SMIN*identity will be used instead of (ca A - w D). If only one singular value is less than SMIN, one element of (ca A - w D) will be perturbed enough to make the smallest singular value roughly SMIN. If both singular values are at least SMIN, (ca A - w D) will not be perturbed. In any case, the perturbation will be at most some small multiple of max(SMIN, ulp*norm(ca A - w D)). The singular values are computed by infinity-norm approximations, and thus will only be correct to a factor of 2 or so.

Note: all input quantities are assumed to be smaller than overflow by a reasonable factor. (See BIGNUM.)

Syntax

Fortran specification:

```
use armpl_library

subroutine slaln2(LTRANS, NA, NW, SMIN, CA, A, LDA, D1, D2, B, LDB, WR, WI, X,
                  LDX, SCALE, XNORM, INFO)
```

C specification:

```
#include "armpl.h"

void slaln2_(const armpl_int_t *ltrans, const armpl_int_t *na,
             const armpl_int_t *nw, const float *smin, const float *ca,
             const float *a, const armpl_int_t *lda, const float *d1,
             const float *d2, const float *b, const armpl_int_t *ldb,
             const float *wr, const float *wi, float *x,
             const armpl_int_t *ldx, float *scale, float *xnorm,
             armpl_int_t *info);
```

Parameters

LTRANS Input parameter.

LTRANS is LOGICAL

=.TRUE.: A-transpose will be used. =.FALSE.: A will be used (not transposed.)

NA Input parameter.

NA is INTEGER

The size of the matrix A. It may (only) be 1 or 2.

NW Input parameter.

NW is INTEGER

1 if "w" is real, 2 if "w" is complex. It may only be 1 or 2.

SMIN Input parameter.

SMIN is REAL

The desired lower bound on the singular values of A. This should be a safe distance away from underflow or overflow, say, between (underflow/machine precision) and (machine precision * overflow). (See BIGNUM and ULP.)

CA Input parameter.

CA is REAL

The coefficient c, which A is multiplied by.

A Input parameter.

A is REAL

A is an array, dimension (LDA, NA). The NA x NA matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of A. It must be at least NA.

D1 Input parameter.

D1 is REAL

The 1,1 element in the diagonal matrix D.

D2 Input parameter.

D2 is REAL

The 2,2 element in the diagonal matrix D. Not used if NA=1.

B Input parameter.

B is REAL

B is an array, dimension (LDB, NW). The NA x NW matrix B (right-hand side). If NW=2 (“w” is complex), column 1 contains the real part of B and column 2 contains the imaginary part.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B. It must be at least NA.

WR Input parameter.

WR is REAL

The real part of the scalar “w”.

WI Input parameter.

WI is REAL

The imaginary part of the scalar “w”. Not used if NW=1.

X Output parameter.

X is REAL

X is an array, dimension (LDX, NW). The NA x NW matrix X (unknowns), as computed by SLALN2. If NW=2 (“w” is complex), on exit, column 1 will contain the real part of X and column 2 will contain the imaginary part.

LDX Input parameter.

LDX is INTEGER

The leading dimension of X. It must be at least NA.

SCALE Output parameter.

SCALE is REAL

The scale factor that B must be multiplied by to insure that overflow does not occur when computing X. Thus, (ca A - w D) X will be SCALE*B, not B (ignoring perturbations of A.) It will be at most 1.

XNORM Output parameter.

XNORM is REAL

The infinity-norm of X, when X is regarded as an NA x NW real matrix.

INFO Output parameter.

INFO is INTEGER

An error flag. It will be set to zero if no error occurs, a negative number if an argument is in error, or a positive number if ca A - w D had to be perturbed. The possible values are: = 0: No error occurred, and (ca A - w D) did not have to be perturbed. = 1: (ca A - w D) had to be perturbed to make its smallest (or only) singular value greater than SMIN. NOTE: In the interests of speed, this routine does not check the inputs for errors.

Related Information

For this routine in other precisions, please see *dlaln2*.

4.17.477 slals0

slals0 applies back the multiplying factors of either the left or the right singular vector matrix of a diagonal matrix appended by a row to the right hand side matrix B in solving the least squares problem using the divide-and-conquer SVD approach.

For the left singular vector matrix, three types of orthogonal matrices are involved:

(1L) Givens rotations: the number of such rotations is GIVPTR; the

```
pairs of columns/rows they were applied to are stored in GIVCOL;
and the C- and S-values of these rotations are stored in GIVNUM.
```

(2L) Permutation. The (NL+1)-st row of B is to be moved to the first

```
row, and for J=2:N, PERM(J)-th row of B is to be moved to the
J-th row.
```

(3L) The left singular vector matrix of the remaining matrix.

For the right singular vector matrix, four types of orthogonal matrices are involved:

(1R) The right singular vector matrix of the remaining matrix.

(2R) If SQRE = 1, one extra Givens rotation to generate the right

```
null space.
```

(3R) The inverse transformation of (2L).

(4R) The inverse transformation of (1L).

Syntax

Fortran specification:

```
use armpl_library

subroutine slals0(ICOMPQ, NL, NR, SQRE, NRHS, B, LDB, BX, LDBX, PERM, GIVPTR,
                  GIVCOL, LDGCOL, GIVNUM, LDGNUM, POLES, DIFL, DIFR, Z, K, C,
                  S, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void slals0_(const armpl_int_t *icompq, const armpl_int_t *nl,
             const armpl_int_t *nr, const armpl_int_t *sqre,
             const armpl_int_t *nrhs, const float *b, const armpl_int_t *ldb,
             float *bx, const armpl_int_t *ldbx, const armpl_int_t *perm,
             const armpl_int_t *givptr, const armpl_int_t *givcol,
             const armpl_int_t *ldgcol, const float *givnum,
             const armpl_int_t *ldgnum, const float *poles, const float *difl,
             const float *difr, const float *z, const armpl_int_t *k,
             const float *c, const float *s, float *work, armpl_int_t *info);
```

Parameters

ICOMPQ Input parameter.

ICOMPQ is INTEGER

Specifies whether singular vectors are to be computed in factored form: = 0: Left singular vector matrix. = 1: Right singular vector matrix.

NL Input parameter.

NL is INTEGER

The row dimension of the upper block. NL >= 1.

NR Input parameter.

NR is INTEGER

The row dimension of the lower block. NR >= 1.

SQRE Input parameter.

SQRE is INTEGER

= 0: the lower block is an NR-by-NR square matrix. = 1: the lower block is an NR-by-(NR+1) rectangular matrix.

The bidiagonal matrix has row dimension N = NL + NR + 1, and column dimension M = N + SQRE.

NRHS Input parameter.

NRHS is INTEGER

The number of columns of B and BX. NRHS must be at least 1.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On input, B contains the right hand sides of the least squares problem in rows 1 through M. On output, B contains the solution X in rows 1 through N.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B. LDB must be at least max(1,MAX(M, N)).

BX Output parameter.

BX is REAL

BX is an array, dimension (LDBX, NRHS) .

LDBX Input parameter.

LDBX is INTEGER

The leading dimension of BX.

PERM Input parameter.

PERM is INTEGER array, dimension (N)

The permutations (from deflation and sorting) applied to the two blocks.

GIVPTR Input parameter.

GIVPTR is INTEGER

The number of Givens rotations which took place in this subproblem.

GIVCOL Input parameter.

GIVCOL is INTEGER array, dimension (LDGCOL, 2)

Each pair of numbers indicates a pair of rows/columns involved in a Givens rotation.

LDGCOL Input parameter.

LDGCOL is INTEGER

The leading dimension of GIVCOL, must be at least N.

GIVNUM Input parameter.

GIVNUM is REAL

GIVNUM is an array, dimension (LDGNUM, 2). Each number indicates the C or S value used in the corresponding Givens rotation.

LDGNUM Input parameter.

LDGNUM is INTEGER

The leading dimension of arrays DIFR, POLES and GIVNUM, must be at least K.

POLES Input parameter.

POLES is REAL

POLES is an array, dimension (LDGNUM, 2). On entry, POLES(1:K, 1) contains the new singular values obtained from solving the secular equation, and POLES(1:K, 2) is an array containing the poles in the secular equation.

DIFL Input parameter.

DIFL is REAL

DIFL is an array, dimension (K).. On entry, DIFL(I) is the distance between I-th updated (undeflated) singular value and the I-th (undeflated) old singular value.

DIFR Input parameter.

DIFR is REAL

DIFR is an array, dimension (LDGNUM, 2).. On entry, DIFR(I, 1) contains the distances between I-th updated (undeflated) singular value and the I+1-th (undeflated) old singular value. And DIFR(I, 2) is the normalizing factor for the I-th right singular vector.

Z Input parameter.

Z is REAL

Z is an array, dimension (K). Contain the components of the deflation-adjusted updating row vector.

K Input parameter.

K is INTEGER

Contains the dimension of the non-deflated matrix, This is the order of the related secular equation. 1 <= K <=N.

C Input parameter.

C is REAL

C contains garbage if SQRE =0 and the C-value of a Givens rotation related to the right null space if SQRE = 1.

S Input parameter.

S is REAL

S contains garbage if SQRE =0 and the S-value of a Givens rotation related to the right null space if SQRE = 1.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (K) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *clals0*, *dlals0* and *zlals0*.

4.17.478 slalsa

slalsa is an itermediate step in solving the least squares problem by computing the SVD of the coefficient matrix in compact form (The singular vectors are computed as products of simple orthorgonal matrices.).

If ICOMPQ = 0, slalsa applies the inverse of the left singular vector matrix of an upper bidiagonal matrix to the right hand side; and if ICOMPQ = 1, slalsa applies the right singular vector matrix to the right hand side. The singular vector matrices were generated in compact form by slalsa.

Syntax

Fortran specification:

```
use armpl_library

subroutine slalsa(ICOMPQ, SMLSIZ, N, NRHS, B, LDB, BX, LDBX, U, LDU, VT, K,
                  DIFL, DIFR, Z, POLES, GIVPTR, GIVCOL, LDGCOL, PERM, GIVNUM,
                  C, S, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void slalsa_(const armpl_int_t *icompq, const armpl_int_t *smlsiz,
             const armpl_int_t *n, const armpl_int_t *nrhs, float *b,
             const armpl_int_t *ldb, float *bx, const armpl_int_t *ldbx,
             const float *u, const armpl_int_t *ldu, const float *vt,
             const armpl_int_t *k, const float *difl, const float *difr,
             const float *z, const float *poles, const armpl_int_t *givptr,
             const armpl_int_t *givcol, const armpl_int_t *ldgcol,
             const armpl_int_t *perm, const float *givnum, const float *c,
             const float *s, float *work, armpl_int_t *iwork,
             armpl_int_t *info);
```

Parameters

ICOMPQ Input parameter.

ICOMPQ is INTEGER

Specifies whether the left or the right singular vector matrix is involved. = 0: Left singular vector matrix = 1: Right singular vector matrix

SMLSIZ Input parameter.

SMLSIZ is INTEGER

The maximum size of the subproblems at the bottom of the computation tree.

N Input parameter.

N is INTEGER

The row and column dimensions of the upper bidiagonal matrix.

NRHS Input parameter.

NRHS is INTEGER

The number of columns of B and BX. NRHS must be at least 1.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On input, B contains the right hand sides of the least squares problem in rows 1 through M. On output, B contains the solution X in rows 1 through N.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B in the calling subprogram. LDB must be at least max(1,MAX(M, N)).

BX Output parameter.

BX is REAL

BX is an array, dimension (LDBX, NRHS). On exit, the result of applying the left or right singular vector matrix to B.

LDBX Input parameter.

LDBX is INTEGER

The leading dimension of BX.

U Input parameter.

U is REAL

U is an array, dimension (LDU, SMLSIZ).. On entry, U contains the left singular vector matrices of all subproblems at the bottom level.

LDU Input parameter.

LDU is INTEGER, LDU = > N.

The leading dimension of arrays U, VT, DIFL, DIFR, POLES, GIVNUM, and Z.

VT Input parameter.

VT is REAL

VT is an array, dimension (LDU, SMLSIZ+1).. On entry, VT^T contains the right singular vector matrices of all subproblems at the bottom level.

K Input parameter.

K is INTEGER array, dimension (N).

DIFL Input parameter.

DIFL is REAL

DIFL is an array, dimension (LDU, NLVL).. where NLVL = INT(log_2 (N/(SMLSIZ+1))) + 1.

DIFR Input parameter.

DIFR is REAL

DIFR is an array, dimension (LDU, 2 * NLVL).. On entry, DIFL(*, I) and DIFR(*, 2 * I -1) record distances between singular values on the I-th level and singular values on the (I -1)-th level, and DIFR(*, 2 * I) record the normalizing factors of the right singular vectors matrices of subproblems on I-th level.

Z Input parameter.

Z is REAL

Z is an array, dimension (LDU, NLVL).. On entry, Z(1, I) contains the components of the deflation- adjusted updating row vector for subproblems on the I-th level.

POLES Input parameter.

POLES is REAL

POLES is an array, dimension (LDU, 2 * NLVL).. On entry, POLES(*, 2 * I -1: 2 * I) contains the new and old singular values involved in the secular equations on the I-th level.

GIVPTR Input parameter.

GIVPTR is INTEGER array, dimension (N).

On entry, GIVPTR(I) records the number of Givens rotations performed on the I-th problem on the computation tree.

GIVCOL Input parameter.

GIVCOL is INTEGER array, dimension (LDGCOL, 2 * NLVL).

On entry, for each I, GIVCOL(*, 2 * I - 1: 2 * I) records the locations of Givens rotations performed on the I-th level on the computation tree.

LDGCOL Input parameter.

LDGCOL is INTEGER, LDGCOL = > N.

The leading dimension of arrays GIVCOL and PERM.

PERM Input parameter.

PERM is INTEGER array, dimension (LDGCOL, NLVL).

On entry, PERM(*, I) records permutations done on the I-th level of the computation tree.

GIVNUM Input parameter.

GIVNUM is REAL

GIVNUM is an array, dimension (LDU, 2 * NLVL).. On entry, GIVNUM(*, 2 *I -1 : 2 * I) records the C- and S- values of Givens rotations performed on the I-th level on the computation tree.

C Input parameter.

C is REAL

C is an array, dimension (N).. On entry, if the I-th subproblem is not square, C(I) contains the C-value of a Givens rotation related to the right null space of the I-th subproblem.

S Input parameter.

S is REAL

S is an array, dimension (N).. On entry, if the I-th subproblem is not square, S(I) contains the S-value of a Givens rotation related to the right null space of the I-th subproblem.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (3*N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *clalsa*, *dlalsa* and *zlalsa*.

4.17.479 slalsd

slalsd uses the singular value decomposition of A to solve the least squares problem of finding X to minimize the Euclidean norm of each column of A*X-B, where A is N-by-N upper bidiagonal, and X and B are N-by-NRHS. The solution X overwrites B.

The singular values of A smaller than RCOND times the largest singular value are treated as zero in solving the least squares problem; in this case a minimum norm solution is returned. The actual singular values are returned in D in ascending order.

This code makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray XMP, Cray YMP, Cray C 90, or Cray 2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine slalsd(UPLO, SMLSIZ, N, NRHS, D, E, B, LDB, RCOND, RANK, WORK,
                  IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void slalsd_(const char *uplo, const armpl_int_t *smlsiz,
             const armpl_int_t *n, const armpl_int_t *nrhs, float *d,
             float *e, float *b, const armpl_int_t *ldb, const float *rcond,
             armpl_int_t *rank, float *work, armpl_int_t *iwork,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': D and E define an upper bidiagonal matrix. = 'L': D and E define a lower bidiagonal matrix.

SMLSIZ Input parameter.

SMLSIZ is INTEGER

The maximum size of the subproblems at the bottom of the computation tree.

N Input parameter.

N is INTEGER

The dimension of the bidiagonal matrix. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of columns of B. NRHS must be at least 1.

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry D contains the main diagonal of the bidiagonal matrix. On exit, if INFO = 0, D contains its singular values.

E Input and output parameter.

E is REAL

E is an array, dimension (N-1). Contains the super-diagonal entries of the bidiagonal matrix. On exit, E has been destroyed.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On input, B contains the right hand sides of the least squares problem. On output, B contains the solution X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B in the calling subprogram. LDB must be at least max(1, N).

RCOND Input parameter.

RCOND is REAL

The singular values of A less than or equal to RCOND times the largest singular value are treated as zero in solving the least squares problem. If RCOND is negative, machine precision is used instead. For example, if diag(S)*X=B were the least squares problem, where diag(S) is a diagonal matrix of singular values, the

solution would be X(i) = B(i) / S(i) if S(i) is greater than RCOND*max(S), and X(i) = 0 if S(i) is less than or equal to RCOND*max(S).

RANK Output parameter.

RANK is INTEGER

The number of singular values of A greater than RCOND times the largest singular value.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension at least. (9* N + 2*N* SMLSIZ + 8*N*NLVL + N* NRHS + (SMLSIZ+1)**2), where NLVL = max(0, INT(log_2 (N/(SMLSIZ+1))) + 1).

IWORK Output parameter.

IWORK is INTEGER array, dimension at least

(3*N*NLVL + 11*N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: The algorithm failed to compute a singular value while working on the submatrix lying in rows and columns INFO/(N+1) through MOD(INFO,N+1).

Related Information

For this routine in other precisions, please see *clalsd*, *dlalsd* and *zlalsd*.

4.17.480 slamrg

slamrg will create a permutation list which will merge the elements of A (which is composed of two independently sorted sets) into a single set which is sorted in ascending order.

Syntax

Fortran specification:

```
use armpl_library

subroutine slamrg(N1, N2, A, STRD1, STRD2, INDEX)
```

C specification:

```
#include "armpl.h"

void slamrg_(const armpl_int_t *n1, const armpl_int_t *n2, const float *a,
             const armpl_int_t *strd1, const armpl_int_t *strd2,
             armpl_int_t *index);
```

Parameters

N1 Input parameter.

N1 is INTEGER

N2 Input parameter.

N2 is INTEGER

These arguments contain the respective lengths of the two sorted lists to be merged.

A Input parameter.

A is REAL

A is an array, dimension (N1+N2). The first N1 elements of A contain a list of numbers which are sorted in either ascending or descending order. Likewise for the final N2 elements.

STRD1 Input parameter.

STRD1 is INTEGER

STRD2 Input parameter.

STRD2 is INTEGER

These are the strides to be taken through the array A. Allowable strides are 1 and -1. They indicate whether a subset of A is sorted in ascending (STRDx = 1) or descending (STRDx = -1) order.

INDEX Output parameter.

INDEX is INTEGER array, dimension (N1+N2)

On exit this array will contain a permutation such that if B(I) = A(INDEX(I)) for I=1,N1+N2, then B will be sorted in ascending order.

Related Information

For this routine in other precisions, please see *dlamrg*.

4.17.481 slamswlq

SLAMSWLQ overwrites the general real M-by-N matrix C with SIDE = 'L' SIDE = 'R' TRANS = 'N': Q * C C * Q TRANS = 'T': Q^T * C C * Q^T where Q is a real orthogonal matrix defined as the product of blocked elementary reflectors computed by short wide LQ factorization (SLASWLQ)

Syntax

Fortran specification:

```
use armpl_library

subroutine slamswlq(SIDE, TRANS, M, N, K, MB, NB, A, LDA, T, LDT, C, LDC,
                    WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void slamswlq_(const char *side, const char *trans, const armpl_int_t *m,
               const armpl_int_t *n, const armpl_int_t *k,
               const armpl_int_t *mb, const armpl_int_t *nb, float *a,
               const armpl_int_t *lda, const float *t, const armpl_int_t *ldt,
               float *c, const armpl_int_t *ldc, float *work,
               const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'T': Transpose, apply Q^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >=0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= M.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. M >= K >= 0;

MB Input parameter.

MB is INTEGER

The row block size to be used in the blocked QR. M >= MB >= 1

NB Input parameter.

NB is INTEGER

The column block size to be used in the blocked QR. NB > M.

NB Input parameter.

NB is INTEGER

The block size to be used in the blocked QR. MB > M.

A Input parameter.

A is REAL

A is an array, dimension. (LDA, M) if SIDE = 'L', (LDA, N) if SIDE = 'R' The i-th row must contain the vector which defines the blocked elementary reflector H(i), for i = 1,2,. . . ,k, as returned by SLASWLQ in the first k rows of its array argument A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDA >= max(1, M); if SIDE = 'R', LDA >= max(1, N).

T Input parameter.

T is REAL

T is an array, dimension. (M * Number of blocks(CEIL(N-K/NB-K)), The blocked upper triangular block reflectors stored in compact form as a sequence of upper triangular blocks. See below for further details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= MB.

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

(workspace) REAL

(workspace) is an array, dimension (MAX(1, LWORK)) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, NB) * MB; if SIDE = 'R', LWORK >= max(1, M) * MB. If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *clamswlq*, *dlamswlq* and *zlamswlq*.

4.17.482 slamtsqr

SLAMTSQR overwrites the general real M-by-N matrix C with SIDE = 'L' SIDE = 'R' TRANS = 'N': Q * C C * Q TRANS = 'T': Q^T * C C * Q^T where Q is a real orthogonal matrix defined as the product of blocked elementary reflectors computed by tall skinny QR factorization (DLATSQR)

Syntax

Fortran specification:

```
use armpl_library

subroutine slamtsqr(SIDE, TRANS, M, N, K, MB, NB, A, LDA, T, LDT, C, LDC,
                    WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void slamtsqr_(const char *side, const char *trans, const armpl_int_t *m,
               const armpl_int_t *n, const armpl_int_t *k,
               const armpl_int_t *mb, const armpl_int_t *nb, float *a,
               const armpl_int_t *lda, const float *t, const armpl_int_t *ldt,
               float *c, const armpl_int_t *ldc, float *work,
               const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left; = 'R': apply Q or Q^T from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'T': Transpose, apply Q^T .

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >=0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. M >= N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. N >= K >= 0;

MB Input parameter.

MB is INTEGER

The block size to be used in the blocked QR. MB > N. (must be the same as DLATSQR)

NB Input parameter.

NB is INTEGER

The column block size to be used in the blocked QR. N >= NB >= 1.

A Input parameter.

A is REAL

A is an array, dimension (LDA, K). The i-th column must contain the vector which defines the blockedelementary reflector H(i), for i = 1,2,...,k, as returned by DLATSQR in the first k columns of its array argument A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDA >= max(1, M); if SIDE = 'R', LDA >= max(1, N).

T Input parameter.

T is REAL

T is an array, dimension. (N * Number of blocks(CEIL(M-K/MB-K)), The blocked upper triangular block reflectors stored in compact form as a sequence of upper triangular blocks. See below for further details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= NB.

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

(workspace) REAL

(workspace) is an array, dimension (MAX(1, LWORK)) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK.

If SIDE = 'L', LWORK >= max(1, N)*NB; if SIDE = 'R', LWORK >= max(1, MB)*NB. If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *clamtsqr*, *dlamtsqr* and *zlamtsqr*.

4.17.483 slaneg

slaneg computes the Sturm count, the number of negative pivots encountered while factoring tridiagonal T - sigma I = L D L^T. This implementation works directly on the factors without forming the tridiagonal matrix T. The Sturm count is also the number of eigenvalues of T less than sigma.

This routine is called from SLARRB.

The current routine does not use the PIVMIN parameter but rather requires IEEE-754 propagation of Infinities and NaNs. This routine also has no input range restrictions but does require default exception handling such that x/0 produces Inf when x is non-zero, and Inf/Inf produces NaN. For more information, see:

```
Marques, Riedy, and Voemel, "Benefits of IEEE-754 Features in
Modern Symmetric Tridiagonal Eigensolvers," SIAM Journal on
Scientific Computing, v28, n5, 2006.  DOI 10.1137/050641624
(Tech report version in LAWN 172 with the same title.)
```

Syntax

Fortran specification:

```
use armpl_library

integer function slaneg(N, D, LLD, SIGMA, PIVMIN, R)
```

C specification:

```
#include "armpl.h"

armpl_int_t slaneg_(const armpl_int_t *n, const float *d, const float *lld,
                    const float *sigma, const float *pivmin,
                    const armpl_int_t *r);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix.

D Input parameter.

D is REAL

D is an array, dimension (N). The N diagonal elements of the diagonal matrix D.

LLD Input parameter.

LLD is REAL

LLD is an array, dimension (N-1). The (N-1) elements L(i)*L(i)*D(i).

SIGMA Input parameter.

SIGMA is REAL

Shift amount in T - sigma I = L D L^T.

PIVMIN Input parameter.

PIVMIN is REAL

The minimum pivot in the Sturm sequence. May be used when zero pivots are encountered on non-IEEE-754 architectures.

R Input parameter.

R is INTEGER

The twist index for the twisted factorization that is used for the negcount.

Related Information

For this routine in other precisions, please see *dlaneg*.

4.17.484 slangb

slangb returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of an n by n band matrix A, with kl sub-diagonals and ku super-diagonals.

Syntax

Fortran specification:

```
use armpl_library

real function slangb(NORM, N, KL, KU, AB, LDAB, WORK)
```

C specification:

```
#include "armpl.h"

float slangb_(const char *norm, const armpl_int_t *n, const armpl_int_t *kl,
              const armpl_int_t *ku, const float *ab, const armpl_int_t *ldab,
              float *work, ... );
```

Returns

SLANGB = (max(abs(A(i,j))), NORM = 'M' or 'm'

((norm1(A), NORM = '1', 'O' or 'o' ((normI(A), NORM = 'I' or 'i' ((normF(A), NORM = 'F', 'f', 'E' or 'e'

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in SLANGB as described above.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0. When N = 0, SLANGB is set to zero.

KL Input parameter.

KL is INTEGER

The number of sub-diagonals of the matrix A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of super-diagonals of the matrix A. KU >= 0.

AB Input parameter.

AB is REAL

AB is an array, dimension (LDAB, N). The band matrix A, stored in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(ku+1+i-j,j) = A(i,j) for max(1,j-ku)<=i<=min(n,j+kl).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KL+KU+1.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= N when NORM = 'I'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *clangb*, *dlangb* and *zlangb*.

4.17.485 slange

`slange` returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a real matrix A.

Syntax

Fortran specification:

```
use armpl_library

real function slange(NORM, M, N, A, LDA, WORK)
```

C specification:

```
#include "armpl.h"

float slange_(const char *norm, const armpl_int_t *m, const armpl_int_t *n,
              const float *a, const armpl_int_t *lda, float *work, ... );
```

Returns

SLANGE = (max(abs(A(i,j))), NORM = ‘M’ or ‘m’

((norm1(A), NORM = ‘1’, ‘O’ or ‘o’ ((normI(A), NORM = ‘I’ or ‘i’ ((normF(A), NORM = ‘F’, ‘f’, ‘E’ or ‘e’

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

`NORM` is `CHARACTER*1`

Specifies the value to be returned in SLANGE as described above.

M Input parameter.

`M` is `INTEGER`

The number of rows of the matrix `A`. `M` >= 0. When `M` = 0, SLANGE is set to zero.

N Input parameter.

`N` is `INTEGER`

The number of columns of the matrix `A`. `N` >= 0. When `N` = 0, SLANGE is set to zero.

A Input parameter.

`A` is `REAL`

`A` is an array, dimension (`LDA`, `N`). The m by n matrix `A`.

LDA Input parameter.

`LDA` is `INTEGER`

The leading dimension of the array `A`. `LDA` >= max(`M`,1).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= M when NORM = 'I'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *clange*, *dlange* and *zlange*. It also exists with a native C interface as *LAPACKE_slange*.

4.17.486 slangt

slangt returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a real tridiagonal matrix A.

Syntax

Fortran specification:

```
use armpl_library

real function slangt(NORM, N, DL, D, DU)
```

C specification:

```
#include "armpl.h"

float slangt_(const char *norm, const armpl_int_t *n, const float *dl,
              const float *d, const float *du, ... );
```

Returns

SLANGT = (max(abs(A(i,j))), NORM = 'M' or 'm'

((norm1(A), NORM = '1', 'O' or 'o' ((normI(A), NORM = 'I' or 'i' ((normF(A), NORM = 'F', 'f', 'E' or 'e'

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in SLANGT as described above.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0. When N = 0, SLANGT is set to zero.

DL Input parameter.

DL is REAL

DL is an array, dimension (N-1). The (n-1) sub-diagonal elements of A.

D Input parameter.

D is REAL

D is an array, dimension (N). The diagonal elements of A.

DU Input parameter.

DU is REAL

DU is an array, dimension (N-1). The (n-1) super-diagonal elements of A.

Related Information

For this routine in other precisions, please see *clangt*, *dlangt* and *zlangt*.

4.17.487 slanhs

slanhs returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a Hessenberg matrix A.

Syntax

Fortran specification:

```
use armpl_library

real function slanhs(NORM, N, A, LDA, WORK)
```

C specification:

```
#include "armpl.h"

float slanhs_(const char *norm, const armpl_int_t *n, const float *a,
              const armpl_int_t *lda, float *work, ... );
```

Returns

SLANHS = (max(abs(A(i,j))), NORM = ‘M’ or ‘m’

((norm1(A), NORM = ‘1’, ‘O’ or ‘o’ ((normI(A), NORM = ‘I’ or ‘i’ ((normF(A), NORM = ‘F’, ‘f’, ‘E’ or ‘e’

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in SLANHS as described above.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0. When N = 0, SLANHS is set to zero.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). The n by n upper Hessenberg matrix A; the part of A below the first sub-diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(N,1).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= N when NORM = 'I'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *clanhs*, *dlanhs* and *zlanhs*.

4.17.488 slansb

slansb returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of an n by n symmetric band matrix A, with k super-diagonals.

Syntax

Fortran specification:

```
use armpl_library

real function slansb(NORM, UPLO, N, K, AB, LDAB, WORK)
```

C specification:

```
#include "armpl.h"

float slansb_(const char *norm, const char *uplo, const armpl_int_t *n,
              const armpl_int_t *k, const float *ab, const armpl_int_t *ldab,
              float *work, ... );
```

Returns

SLANSB = (max(abs(A(i,j))), NORM = 'M' or 'm'

((norm1(A), NORM = '1', 'O' or 'o' ((normI(A), NORM = 'I' or 'i' ((normF(A), NORM = 'F', 'f', 'E' or 'e'

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in SLANSB as described above.

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the band matrix A is supplied. = 'U': Upper triangular part is supplied = 'L': Lower triangular part is supplied

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0. When N = 0, SLANSB is set to zero.

K Input parameter.

K is INTEGER

The number of super-diagonals or sub-diagonals of the band matrix A. K >= 0.

AB Input parameter.

AB is REAL

AB is an array, dimension (LDAB, N). The upper or lower triangle of the symmetric band matrix A, stored in the first K+1 rows of AB. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(k+1+i-j,j) = A(i,j) for max(1,j-k)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+k).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= K+1.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= N when NORM = 'I' or '1' or 'O'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *clansb*, *dlansb* and *zlansb*.

4.17.489 slansf

slansf returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a real symmetric matrix A in RFP format.

Syntax

Fortran specification:

```
use armpl_library

real function slansf(NORM, TRANSR, UPLO, N, A, WORK)
```

C specification:

```
#include "armpl.h"

float slansf_(const char *norm, const char *transr, const char *uplo,
              const armpl_int_t *n, const float *a, float *work, ... );
```

Returns

SLANSF = (max(abs(A(i,j))), NORM = ‘M’ or ‘m’

((norm1(A), NORM = ‘1’, ‘O’ or ‘o’ ((normI(A), NORM = ‘I’ or ‘i’ ((normF(A), NORM = ‘F’, ‘f’, ‘E’ or ‘e’

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in SLANSF as described above.

TRANSR Input parameter.

TRANSR is CHARACTER*1

Specifies whether the RFP format of A is normal or transposed format. = ‘N’: RFP format is Normal; = ‘T’: RFP format is Transpose.

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the RFP matrix A came from an upper or lower triangular matrix as follows: = ‘U’: RFP A came from an upper triangular matrix; = ‘L’: RFP A came from a lower triangular matrix.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0. When N = 0, SLANSF is set to zero.

A Input parameter.

A is REAL

A is an array, dimension (N*(N+1)/2);. On entry, the upper (if UPLO = ‘U’) or lower (if UPLO = ‘L’) part of the symmetric matrix A stored in RFP format. See the “Notes” below for more details. Unchanged on exit.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= N when NORM = ‘I’ or ‘1’ or ‘O’; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *dlansf*.

4.17.490 slansp

`slansp` returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a real symmetric matrix A, supplied in packed form.

Syntax

Fortran specification:

```
use armpl_library

real function slansp(NORM, UPLO, N, AP, WORK)
```

C specification:

```
#include "armpl.h"

float slansp_(const char *norm, const char *uplo, const armpl_int_t *n,
              const float *ap, float *work, ... );
```

Returns

SLANSP = (max(abs(A(i,j))), NORM = ‘M’ or ‘m’

((norm1(A), NORM = ‘1’, ‘O’ or ‘o’ ((normI(A), NORM = ‘I’ or ‘i’ ((normF(A), NORM = ‘F’, ‘f’, ‘E’ or ‘e’

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

`NORM` is `CHARACTER*1`

Specifies the value to be returned in SLANSP as described above.

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

Specifies whether the upper or lower triangular part of the symmetric matrix A is supplied. = ‘`U`’: Upper triangular part of A is supplied = ‘`L`’: Lower triangular part of A is supplied

N Input parameter.

`N` is `INTEGER`

The order of the matrix A. `N` >= 0. When `N` = 0, SLANSP is set to zero.

AP Input parameter.

`AP` is `REAL`

`AP` is an array, dimension (N*(N+1)/2). The upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array `AP` as follows: if `UPLO` = ‘`U`’, AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if `UPLO` = ‘`L`’, AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

WORK Output parameter.

`WORK` is `REAL`

`WORK` is an array, dimension (MAX(1,LWORK)),. where LWORK >= `N` when `NORM` = ‘I’ or ‘1’ or ‘O’; otherwise, `WORK` is not referenced.

Related Information

For this routine in other precisions, please see *clansp*, *dlansp* and *zlansp*.

4.17.491 slanst

`slanst` returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a real symmetric tridiagonal matrix A.

Syntax

Fortran specification:

```
use armpl_library

real function slanst(NORM, N, D, E)
```

C specification:

```
#include "armpl.h"

float slanst_(const char *norm, const armpl_int_t *n, const float *d,
              const float *e, ... );
```

Returns

SLANST = (max(abs(A(i,j))), NORM = 'M' or 'm'

((norm1(A), NORM = '1', 'O' or 'o' ((normI(A), NORM = 'I' or 'i' ((normF(A), NORM = 'F', 'f', 'E' or 'e'

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

`NORM` is `CHARACTER*1`

Specifies the value to be returned in SLANST as described above.

N Input parameter.

`N` is `INTEGER`

The order of the matrix A. `N` >= 0. When `N` = 0, SLANST is set to zero.

D Input parameter.

`D` is `REAL`

`D` is an array, dimension (`N`). The diagonal elements of A.

E Input parameter.

`E` is `REAL`

`E` is an array, dimension (N-1). The (n-1) sub-diagonal or super-diagonal elements of A.

Related Information

For this routine in other precisions, please see *dlanst*.

4.17.492 slansy

`slansy` returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a real symmetric matrix A.

Syntax

Fortran specification:

```
use armpl_library

real function slansy(NORM, UPLO, N, A, LDA, WORK)
```

C specification:

```
#include "armpl.h"

float slansy_(const char *norm, const char *uplo, const armpl_int_t *n,
              const float *a, const armpl_int_t *lda, float *work, ... );
```

Returns

SLANSY = (max(abs(A(i,j))), NORM = ‘M’ or ‘m’

((norm1(A), NORM = ‘1’, ‘O’ or ‘o’ ((normI(A), NORM = ‘I’ or ‘i’ ((normF(A), NORM = ‘F’, ‘f’, ‘E’ or ‘e’

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

`NORM` is `CHARACTER*1`

Specifies the value to be returned in SLANSY as described above.

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

Specifies whether the upper or lower triangular part of the symmetric matrix `A` is to be referenced. = ‘`U`’: Upper triangular part of `A` is referenced = ‘`L`’: Lower triangular part of `A` is referenced

N Input parameter.

`N` is `INTEGER`

The order of the matrix `A`. `N` >= 0. When `N` = 0, SLANSY is set to zero.

A Input parameter.

`A` is `REAL`

`A` is an array, dimension (`LDA`, `N`). The symmetric matrix `A`. If `UPLO` = ‘`U`’, the leading n by n upper triangular part of `A` contains the upper triangular part of the matrix `A`, and the strictly lower triangular part of A is not

referenced. If UPLO = 'L', the leading n by n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(N,1).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= N when NORM = 'I' or '1' or 'O'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *clansy*, *dlansy* and *zlansy*. It also exists with a native C interface as *LAPACKE_slansy*.

4.17.493 slantb

slantb returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of an n by n triangular band matrix A, with (k + 1) diagonals.

Syntax

Fortran specification:

```
use armpl_library

real function slantb(NORM, UPLO, DIAG, N, K, AB, LDAB, WORK)
```

C specification:

```
#include "armpl.h"

float slantb_(const char *norm, const char *uplo, const char *diag,
              const armpl_int_t *n, const armpl_int_t *k, const float *ab,
              const armpl_int_t *ldab, float *work, ... );
```

Returns

SLANTB = (max(abs(A(i,j))), NORM = 'M' or 'm'

((norm1(A), NORM = '1', 'O' or 'o' ((normI(A), NORM = 'I' or 'i' ((normF(A), NORM = 'F', 'f', 'E' or 'e'

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in SLANTB as described above.

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the matrix A is upper or lower triangular. = 'U': Upper triangular = 'L': Lower triangular

DIAG Input parameter.

DIAG is CHARACTER*1

Specifies whether or not the matrix A is unit triangular. = 'N': Non-unit triangular = 'U': Unit triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0. When N = 0, SLANTB is set to zero.

K Input parameter.

K is INTEGER

The number of super-diagonals of the matrix A if UPLO = 'U', or the number of sub-diagonals of the matrix A if UPLO = 'L'. K >= 0.

AB Input parameter.

AB is REAL

AB is an array, dimension (LDAB, N). The upper or lower triangular band matrix A, stored in the first k+1 rows of AB. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(k+1+i-j,j) = A(i,j) for max(1,j-k)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+k). Note that when DIAG = 'U', the elements of the array AB corresponding to the diagonal elements of the matrix A are not referenced, but are assumed to be one.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= K+1.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= N when NORM = 'I'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *clantb*, *dlantb* and *zlantb*.

4.17.494 slantp

slantp returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a triangular matrix A, supplied in packed form.

Syntax

Fortran specification:

```
use armpl_library

real function slantp(NORM, UPLO, DIAG, N, AP, WORK)
```

C specification:

```
#include "armpl.h"

float slantp_(const char *norm, const char *uplo, const char *diag,
              const armpl_int_t *n, const float *ap, float *work, ... );
```

Returns

SLANTP = (max(abs(A(i,j))), NORM = ‘M’ or ‘m’

((norm1(A), NORM = ‘1’, ‘O’ or ‘o’ ((normI(A), NORM = ‘I’ or ‘i’ ((normF(A), NORM = ‘F’, ‘f’, ‘E’ or ‘e’

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in SLANTP as described above.

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the matrix A is upper or lower triangular. = ‘U’: Upper triangular = ‘L’: Lower triangular

DIAG Input parameter.

DIAG is CHARACTER*1

Specifies whether or not the matrix A is unit triangular. = ‘N’: Non-unit triangular = ‘U’: Unit triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0. When N = 0, SLANTP is set to zero.

AP Input parameter.

AP is REAL

AP is an array, dimension (N*(N+1)/2). The upper or lower triangular matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = ‘U’, AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = ‘L’, AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n. Note that when DIAG = ‘U’, the elements of the array AP corresponding to the diagonal elements of the matrix A are not referenced, but are assumed to be one.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= N when NORM = ‘I’; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *clantp*, *dlantp* and *zlantp*.

4.17.495 slantr

`slantr` returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a trapezoidal or triangular matrix A.

Syntax

Fortran specification:

```
use armpl_library

real function slantr(NORM, UPLO, DIAG, M, N, A, LDA, WORK)
```

C specification:

```
#include "armpl.h"

float slantr_(const char *norm, const char *uplo, const char *diag,
              const armpl_int_t *m, const armpl_int_t *n, const float *a,
              const armpl_int_t *lda, float *work, ... );
```

Returns

SLANTR = (max(abs(A(i,j))), NORM = ‘M’ or ‘m’

((norm1(A), NORM = ‘1’, ‘O’ or ‘o’ ((normI(A), NORM = ‘I’ or ‘i’ ((normF(A), NORM = ‘F’, ‘f’, ‘E’ or ‘e’

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

`NORM` is `CHARACTER*1`

Specifies the value to be returned in SLANTR as described above.

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

Specifies whether the matrix `A` is upper or lower trapezoidal. = ‘`U`’: Upper trapezoidal = ‘`L`’: Lower trapezoidal Note that `A` is triangular instead of trapezoidal if `M` = `N`.

DIAG Input parameter.

`DIAG` is `CHARACTER*1`

Specifies whether or not the matrix `A` has unit diagonal. = ‘`N`’: Non-unit diagonal = ‘`U`’: Unit diagonal

M Input parameter.

`M` is `INTEGER`

The number of rows of the matrix `A`. `M` >= 0, and if `UPLO` = ‘`U`’, `M` <= `N`. When `M` = 0, SLANTR is set to zero.

N Input parameter.

`N` is `INTEGER`

The number of columns of the matrix `A`. `N` >= 0, and if `UPLO` = ‘`L`’, `N` <= `M`. When `N` = 0, SLANTR is set to zero.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). The trapezoidal matrix A (A is triangular if M = N). If UPLO = 'U', the leading m by n upper trapezoidal part of the array A contains the upper trapezoidal matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading m by n lower trapezoidal part of the array A contains the lower trapezoidal matrix, and the strictly upper triangular part of A is not referenced. Note that when DIAG = 'U', the diagonal elements of A are not referenced and are assumed to be one.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(M,1).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= M when NORM = 'I'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *clantr*, *dlantr* and *zlantr*. It also exists with a native C interface as *LAPACKE_slantr*.

4.17.496 slanv2

slanv2 computes the Schur factorization of a real 2-by-2 nonsymmetric matrix in standard form:

```
[ A   B ] = [ CS -SN ] [ AA  BB ] [ CS  SN ]
[ C   D ]   [ SN  CS ] [ CC  DD ] [-SN  CS ]
```

where either 1) CC = 0 so that AA and DD are real eigenvalues of the matrix, or 2) AA = DD and BB*CC < 0, so that AA + or - sqrt(BB*CC) are complex conjugate eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine slanv2(A, B, C, D, RT1R, RT1I, RT2R, RT2I, CS, SN)
```

C specification:

```
#include "armpl.h"

void slanv2_(float *a, float *b, float *c, float *d, float *rt1r, float *rt1i,
             float *rt2r, float *rt2i, float *cs, float *sn);
```

Parameters

A Input and output parameter.

A is REAL

B Input and output parameter.

B is REAL

C Input and output parameter.

C is REAL

D Input and output parameter.

D is REAL

On entry, the elements of the input matrix. On exit, they are overwritten by the elements of the standardised Schur form.

RT1R Output parameter.

RT1R is REAL

RT1I Output parameter.

RT1I is REAL

RT2R Output parameter.

RT2R is REAL

RT2I Output parameter.

RT2I is REAL

The real and imaginary parts of the eigenvalues. If the eigenvalues are a complex conjugate pair, RT1I > 0.

CS Output parameter.

CS is REAL

SN Output parameter.

SN is REAL

Parameters of the rotation matrix.

Related Information

For this routine in other precisions, please see *dlanv2*.

4.17.497 slapll

Given two column vectors X and Y, let

```
A = ( X Y ).
```

The subroutine first computes the QR factorization of A = Q*R, and then computes the SVD of the 2-by-2 upper triangular matrix R. The smaller singular value of R is returned in SSMIN, which is used as the measurement of the linear dependency of the vectors X and Y.

Syntax

Fortran specification:

```
use armpl_library

subroutine slapll(N, X, INCX, Y, INCY, SSMIN)
```

C specification:

```
#include "armpl.h"

void slapll_(const armpl_int_t *n, float *x, const armpl_int_t *incx,
             float *y, const armpl_int_t *incy, float *ssmin);
```

Parameters

N Input parameter.

N is INTEGER

The length of the vectors X and Y.

X Input and output parameter.

X is REAL

X is an array,. dimension (1+(N-1)*INCX) On entry, X contains the N-vector X. On exit, X is overwritten.

INCX Input parameter.

INCX is INTEGER

The increment between successive elements of X. INCX > 0.

Y Input and output parameter.

Y is REAL

Y is an array,. dimension (1+(N-1)*INCY) On entry, Y contains the N-vector Y. On exit, Y is overwritten.

INCY Input parameter.

INCY is INTEGER

The increment between successive elements of Y. INCY > 0.

SSMIN Output parameter.

SSMIN is REAL

The smallest singular value of the N-by-2 matrix A = (X Y).

Related Information

For this routine in other precisions, please see *clapll*, *dlapll* and *zlapll*.

4.17.498 slapmr

slapmr rearranges the rows of the M by N matrix X as specified by the permutation K(1),K(2),...,K(M) of the integers 1,...,M. If FORWRD = .TRUE., forward permutation:

```
X(K(I),*) is moved X(I,*) for I = 1,2,...,M.
```

If FORWRD = .FALSE., backward permutation:

```
X(I,*) is moved to X(K(I),*) for I = 1,2,...,M.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine slapmr(FORWRD, M, N, X, LDX, K)
```

C specification:

```
#include "armpl.h"

void slapmr_(const armpl_int_t *forwrd, const armpl_int_t *m,
             const armpl_int_t *n, float *x, const armpl_int_t *ldx,
             armpl_int_t *k);
```

Parameters

FORWRD Input parameter.

FORWRD is LOGICAL

= .TRUE., forward permutation = .FALSE., backward permutation

M Input parameter.

M is INTEGER

The number of rows of the matrix X. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix X. N >= 0.

X Input and output parameter.

X is REAL

X is an array, dimension (LDX, N). On entry, the M by N matrix X. On exit, X contains the permuted matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X, LDX >= MAX(1, M).

K Input and output parameter.

K is INTEGER array, dimension (M)

On entry, K contains the permutation vector. K is used as internal workspace, but reset to its original value on output.

Related Information

For this routine in other precisions, please see *clapmr*, *dlapmr* and *zlapmr*. It also exists with a native C interface as *LAPACKE_slapmr*.

4.17.499 slapmt

slapmt rearranges the columns of the M by N matrix X as specified by the permutation K(1),K(2),...,K(N) of the integers 1,...,N. If FORWRD = .TRUE., forward permutation:

```
X(*,K(J)) is moved X(*,J) for J = 1,2,...,N.
```

If FORWRD = .FALSE., backward permutation:

```
X(*,J) is moved to X(*,K(J)) for J = 1,2,...,N.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine slapmt(FORWRD, M, N, X, LDX, K)
```

C specification:

```
#include "armpl.h"

void slapmt_(const armpl_int_t *forwrd, const armpl_int_t *m,
             const armpl_int_t *n, float *x, const armpl_int_t *ldx,
             armpl_int_t *k);
```

Parameters

FORWRD Input parameter.

FORWRD is LOGICAL

= .TRUE., forward permutation = .FALSE., backward permutation

M Input parameter.

M is INTEGER

The number of rows of the matrix X. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix X. N >= 0.

X Input and output parameter.

X is REAL

X is an array, dimension (LDX, N). On entry, the M by N matrix X. On exit, X contains the permuted matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X, LDX >= MAX(1, M).

K Input and output parameter.

K is INTEGER array, dimension (N)

On entry, K contains the permutation vector. K is used as internal workspace, but reset to its original value on output.

Related Information

For this routine in other precisions, please see *clapmt*, *dlapmt* and *zlapmt*. It also exists with a native C interface as *LAPACKE_slapmt*.

4.17.500 slapy2

`slapy2` returns sqrt(x**2+y**2), taking care not to cause unnecessary overflow.

Syntax

Fortran specification:

```
use armpl_library

real function slapy2(X, Y)
```

C specification:

```
#include "armpl.h"

float slapy2_(const float *x, const float *y);
```

Parameters

X Input parameter.

X is REAL

Y Input parameter.

Y is REAL

X and Y specify the values x and y.

Related Information

For this routine in other precisions, please see *dlapy2*. It also exists with a native C interface as *LAPACKE_slapy2*.

4.17.501 slapy3

`slapy3` returns sqrt(x**2+y**2+z**2), taking care not to cause unnecessary overflow.

Syntax

Fortran specification:

```
use armpl_library

real function slapy3(X, Y, Z)
```

C specification:

```
#include "armpl.h"

float slapy3_(const float *x, const float *y, const float *z);
```

Parameters

X Input parameter.

X is REAL

Y Input parameter.

Y is REAL

Z Input parameter.

Z is REAL

X, Y and Z specify the values x, y and z.

Related Information

For this routine in other precisions, please see *dlapy3*. It also exists with a native C interface as *LAPACKE_slapy3*.

4.17.502 slaqgb

slaqgb equilibrates a general M by N band matrix A with KL subdiagonals and KU superdiagonals using the row and scaling factors in the vectors R and C.

Syntax

Fortran specification:

```
use armpl_library

subroutine slaqgb(M, N, KL, KU, AB, LDAB, R, C, ROWCND, COLCND, AMAX, EQUED)
```

C specification:

```
#include "armpl.h"

void slaqgb_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *kl, const armpl_int_t *ku, float *ab,
             const armpl_int_t *ldab, const float *r, const float *c,
             const float *rowcnd, const float *colcnd, const float *amax,
             char *equed, ... );
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

AB Input and output parameter.

AB is REAL

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(ku+1+i-j,j) = A(i,j) for max(1,j-ku)<=i<=min(m,j+kl)

On exit, the equilibrated matrix, in the same storage format as A. See EQUED for the form of the equilibrated matrix.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDA >= KL+KU+1.

R Input parameter.

R is REAL

R is an array, dimension (M). The row scale factors for A.

C Input parameter.

C is REAL

C is an array, dimension (N). The column scale factors for A.

ROWCND Input parameter.

ROWCND is REAL

Ratio of the smallest R(i) to the largest R(i).

COLCND Input parameter.

COLCND is REAL

Ratio of the smallest C(i) to the largest C(i).

AMAX Input parameter.

AMAX is REAL

Absolute value of largest matrix entry.

EQUED Output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration = 'R': Row equilibration, i.e., A has been premultiplied by diag(R). = 'C': Column equilibration, i.e., A has been postmultiplied by diag(C). = 'B': Both row and column equilibration, i.e., A has been replaced by diag(R) * A * diag(C).

Related Information

For this routine in other precisions, please see *claqgb*, *dlaqgb* and *zlaqgb*.

4.17.503 slaqge

slaqge equilibrates a general M by N matrix A using the row and column scaling factors in the vectors R and C.

Syntax

Fortran specification:

```
use armpl_library

subroutine slaqge(M, N, A, LDA, R, C, ROWCND, COLCND, AMAX, EQUED)
```

C specification:

```
#include "armpl.h"

void slaqge_(const armpl_int_t *m, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, const float *r, const float *c,
             const float *rowcnd, const float *colcnd, const float *amax,
             char *equed, ... );
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the M by N matrix A. On exit, the equilibrated matrix. See EQUED for the form of the equilibrated matrix.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(M,1).

R Input parameter.

R is REAL

R is an array, dimension (M). The row scale factors for A.

C Input parameter.

C is REAL

C is an array, dimension (N). The column scale factors for A.

ROWCND Input parameter.

ROWCND is REAL

Ratio of the smallest R(i) to the largest R(i).

COLCND Input parameter.

COLCND is REAL

Ratio of the smallest C(i) to the largest C(i).

AMAX Input parameter.

AMAX is REAL

Absolute value of largest matrix entry.

EQUED Output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration = 'R': Row equilibration, i.e., A has been premultiplied by diag(R). = 'C': Column equilibration, i.e., A has been postmultiplied by diag(C). = 'B': Both row and column equilibration, i.e., A has been replaced by diag(R) * A * diag(C).

Related Information

For this routine in other precisions, please see *claqge*, *dlaqge* and *zlaqge*.

4.17.504 slaqp2

slaqp2 computes a QR factorization with column pivoting of the block A(OFFSET+1:M,1:N). The block A(1:OFFSET,1:N) is accordingly pivoted, but not factorized.

Syntax

Fortran specification:

```
use armpl_library

subroutine slaqp2(M, N, OFFSET, A, LDA, JPVT, TAU, VN1, VN2, WORK)
```

C specification:

```
#include "armpl.h"

void slaqp2_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *offset, float *a, const armpl_int_t *lda,
             armpl_int_t *jpvt, float *tau, float *vn1, float *vn2,
             float *work);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

OFFSET Input parameter.

OFFSET is INTEGER

The number of rows of the matrix A that must be pivoted but no factorized. OFFSET >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the upper triangle of block A(OFFSET+1:M,1:N) is the triangular factor obtained; the elements in block A(OFFSET+1:M,1:N) below the diagonal, together with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors. Block A(1:OFFSET,1:N) has been accordingly pivoted, but no factorized.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

JPVT Input and output parameter.

JPVT is INTEGER array, dimension (N)

On entry, if JPVT(i) .ne. 0, the i-th column of A is permuted to the front of A*P (a leading column); if JPVT(i) = 0, the i-th column of A is a free column. On exit, if JPVT(i) = k, then the i-th column of A*P was the k-th column of A.

TAU Output parameter.

TAU is REAL

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors.

VN1 Input and output parameter.

VN1 is REAL

VN1 is an array, dimension (N). The vector with the partial column norms.

VN2 Input and output parameter.

VN2 is REAL

VN2 is an array, dimension (N). The vector with the exact column norms.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (N) .

Related Information

For this routine in other precisions, please see *claqp2*, *dlaqp2* and *zlaqp2*.

4.17.505 slaqps

slaqps computes a step of QR factorization with column pivoting of a real M-by-N matrix A by using Blas-3. It tries to factorize NB columns from A starting from the row OFFSET+1, and updates all of the matrix with Blas-3 xGEMM.

In some cases, due to catastrophic cancellations, it cannot factorize NB columns. Hence, the actual number of factorized columns is returned in KB.

Block A(1:OFFSET,1:N) is accordingly pivoted, but not factorized.

Syntax

Fortran specification:

```
use armpl_library

subroutine slaqps(M, N, OFFSET, NB, KB, A, LDA, JPVT, TAU, VN1, VN2, AUXV, F,
                  LDF)
```

C specification:

```
#include "armpl.h"

void slaqps_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *offset, const armpl_int_t *nb,
             armpl_int_t *kb, float *a, const armpl_int_t *lda,
             armpl_int_t *jpvt, float *tau, float *vn1, float *vn2,
             float *auxv, float *f, const armpl_int_t *ldf);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0

OFFSET Input parameter.

OFFSET is INTEGER

The number of rows of A that have been factorized in previous steps.

NB Input parameter.

NB is INTEGER

The number of columns to factorize.

KB Output parameter.

KB is INTEGER

The number of columns actually factorized.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, block A(OFFSET+1:M,1:KB) is the triangular factor obtained and block A(1:OFFSET,1:N) has been accordingly pivoted, but no factorized. The rest of the matrix, block A(OFFSET+1:M,KB+1:N) has been updated.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

JPVT Input and output parameter.

JPVT is INTEGER array, dimension (N)

JPVT(I) = K <==> Column K of the full matrix A has been permuted into position I in AP.

TAU Output parameter.

TAU is REAL

TAU is an array, dimension (KB). The scalar factors of the elementary reflectors.

VN1 Input and output parameter.

VN1 is REAL

VN1 is an array, dimension (N). The vector with the partial column norms.

VN2 Input and output parameter.

VN2 is REAL

VN2 is an array, dimension (N). The vector with the exact column norms.

AUXV Input and output parameter.

AUXV is REAL

AUXV is an array, dimension (NB). Auxiliar vector.

F Input and output parameter.

F is REAL

F is an array, dimension (LDF, NB). Matrix $F^T = L*Y^T *A$.

LDF Input parameter.

LDF is INTEGER

The leading dimension of the array F. LDF >= max(1, N).

Related Information

For this routine in other precisions, please see *claqps*, *dlaqps* and *zlaqps*.

4.17.506 slaqr0

```
SLAQR0 computes the eigenvalues of a Hessenberg matrix H
   and, optionally, the matrices T and Z from the Schur decomposition
   H = Z T Z**T, where T is an upper quasi-triangular matrix (the
   Schur form), and Z is the orthogonal matrix of Schur vectors.

   Optionally Z may be postmultiplied into an input orthogonal
   matrix Q so that this routine can give the Schur factorization
   of a matrix A which has been reduced to the Hessenberg form H
   by the orthogonal matrix Q:  A = Q*H*Q**T = (QZ)*T*(QZ)**T.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine slaqr0(WANTT, WANTZ, N, ILO, IHI, H, LDH, WR, WI, ILOZ, IHIZ, Z,
                  LDZ, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void slaqr0_(const armpl_int_t *wantt, const armpl_int_t *wantz,
             const armpl_int_t *n, const armpl_int_t *ilo,
             const armpl_int_t *ihi, float *h, const armpl_int_t *ldh,
             float *wr, float *wi, const armpl_int_t *iloz,
             const armpl_int_t *ihiz, float *z, const armpl_int_t *ldz,
             float *work, const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

WANTT Input parameter.

WANTT is LOGICAL

= .TRUE. : the full Schur form T is required; = .FALSE.: only eigenvalues are required.

WANTZ Input parameter.

WANTZ is LOGICAL

= .TRUE. : the matrix of Schur vectors Z is required; = .FALSE.: Schur vectors are not required.

N Input parameter.

N is INTEGER

The order of the matrix H. N .GE. 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

It is assumed that H is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N and, if ILO.GT.1, H(ILO,ILO-1) is zero. ILO and IHI are normally set by a previous call to SGEBAL, and then passed to SGEHRD when the matrix output by SGEBAL is reduced to Hessenberg form. Otherwise, ILO and IHI should be set to 1 and N, respectively. If N.GT.0, then 1.LE.ILO.LE.IHI.LE.N. If N = 0, then ILO = 1 and IHI = 0.

H Input and output parameter.

H is REAL

H is an array, dimension (LDH, N). On entry, the upper Hessenberg matrix H. On exit, if INFO = 0 and WANTT is .TRUE., then H contains the upper quasi-triangular matrix T from the Schur decomposition (the Schur form); 2-by-2 diagonal blocks (corresponding to complex conjugate pairs of eigenvalues) are returned in standard form, with H(i,i) = H(i+1,i+1) and H(i+1,i)*H(i,i+1).LT.0. If INFO = 0 and WANTT is .FALSE., then the contents of H are unspecified on exit. (The output value of H when INFO.GT.0 is given under the description of INFO below.)

This subroutine may explicitly set H(i,j) = 0 for i.GT.j and j = 1, 2, ... ILO-1 or j = IHI+1, IHI+2, ... N.

LDH Input parameter.

LDH is INTEGER

The leading dimension of the array H. LDH .GE. max(1, N).

WR Output parameter.

WR is REAL

WR is an array, dimension (IHI) .

WI Output parameter.

WI is REAL

WI is an array, dimension (IHI). The real and imaginary parts, respectively, of the computed eigenvalues of H(ILO:IHI,ILO:IHI) are stored in WR(ILO:IHI) and WI(ILO:IHI). If two eigenvalues are computed as a complex conjugate pair, they are stored in consecutive elements of WR and WI, say the i-th and (i+1)th, with WI(i) .GT. 0 and WI(i+1) .LT. 0. If WANTT is .TRUE., then the eigenvalues are stored in the same order as on the diagonal of the Schur form returned in H, with WR(i) = H(i,i) and, if H(i:i+1,i:i+1) is a 2-by-2 diagonal block, WI(i) = sqrt(-H(i+1,i)*H(i,i+1)) and WI(i+1) = -WI(i).

ILOZ Input parameter.

ILOZ is INTEGER

IHIZ Input parameter.

IHIZ is INTEGER

Specify the rows of Z to which transformations must be applied if WANTZ is .TRUE.. 1 .LE. ILOZ .LE. ILO; IHI .LE. IHIZ .LE. N.

Z Input and output parameter.

Z is REAL

Z is an array, dimension (LDZ, IHI). If WANTZ is .FALSE., then Z is not referenced. If WANTZ is .TRUE., then Z(ILO:IHI,ILOZ:IHIZ) is replaced by Z(ILO:IHI,ILOZ:IHIZ)*U where U is the orthogonal Schur factor of H(ILO:IHI,ILO:IHI). (The output value of Z when INFO.GT.0 is given under the description of INFO below.)

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. if WANTZ is .TRUE. then LDZ.GE.MAX(1, IHIZ). Otherwize, LDZ.GE.1.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension LWORK. On exit, if LWORK = -1, WORK(1) returns an estimate of the optimal value for LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK .GE. max(1, N) is sufficient, but LWORK typically as large as 6* N may be required for optimal performance. A workspace query to determine the optimal workspace size is recommended.

If LWORK = -1, then SLAQR0 does a workspace query. In this case, SLAQR0 checks the input parameters and estimates the optimal workspace size for the given values of N, ILO and IHI. The estimate is returned in WORK(1). No error message related to LWORK is issued by XERBLA. Neither H nor Z are accessed.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit .GT. 0: if INFO = i, SLAQR0 failed to compute all of the eigenvalues. Elements 1:ilo-1 and i+1:n of WR and WI contain those eigenvalues which have been successfully computed. (Failures are rare.)

If INFO .GT. 0 and WANT is .FALSE., then on exit, the remaining unconverged eigenvalues are the eigenvalues of the upper Hessenberg matrix rows and columns ILO through INFO of the final, output value of H.

If INFO .GT. 0 and WANTT is .TRUE., then on exit

(*) (initial value of H)*U = U*(final value of H)

where U is an orthogonal matrix. The final value of H is upper Hessenberg and quasi-triangular in rows and columns INFO+1 through IHI.

If INFO .GT. 0 and WANTZ is .TRUE., then on exit

(final value of Z(ILO:IHI,ILOZ:IHIZ) = (initial value of Z(ILO:IHI,ILOZ:IHIZ)*U

where U is the orthogonal matrix in (*) (regard- less of the value of WANTT.)

If INFO .GT. 0 and WANTZ is .FALSE., then Z is not accessed.

Related Information

For this routine in other precisions, please see *claqr0*, *dlaqr0* and *zlaqr0*.

4.17.507 slaqr1

```
Given a 2-by-2 or 3-by-3 matrix H, SLAQR1 sets v to a
     scalar multiple of the first column of the product

     (*)  K = (H - (sr1 + i*si1)*I)*(H - (sr2 + i*si2)*I)

     scaling to avoid overflows and most underflows. It
     is assumed that either

               1) sr1 = sr2 and si1 = -si2
          or
               2) si1 = si2 = 0.

     This is useful for starting double implicit shift bulges
     in the QR algorithm.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine slaqr1(N, H, LDH, SR1, SI1, SR2, SI2, V)
```

C specification:

```
#include "armpl.h"

void slaqr1_(const armpl_int_t *n, const float *h, const armpl_int_t *ldh,
             const float *sr1, float *si1, float *sr2, float *si2, float *v);
```

Parameters

N Input parameter.

N is INTEGER

Order of the matrix H. N must be either 2 or 3.

H Input parameter.

H is REAL

H is an array, dimension (LDH, N). The 2-by-2 or 3-by-3 matrix H in (*).

LDH Input parameter.

LDH is INTEGER

The leading dimension of H as declared in the calling procedure. LDH.GE.N

SR1 Input parameter.

SR1 is REAL

SI1 Input parameter.

SI1 is REAL

SR2 Input parameter.

SR2 is REAL

SI2 Input parameter.

SI2 is REAL

The shifts in (*).

V Output parameter.

V is REAL

V is an array, dimension (N). A scalar multiple of the first column of the matrix K in (*).

Related Information

For this routine in other precisions, please see *claqr1*, *dlaqr1* and *zlaqr1*.

4.17.508 slaqr2

```
SLAQR2 is identical to SLAQR3 except that it avoids
   recursion by calling SLAHQR instead of SLAQR4.

   Aggressive early deflation:

   This subroutine accepts as input an upper Hessenberg matrix
   H and performs an orthogonal similarity transformation
   designed to detect and deflate fully converged eigenvalues from
   a trailing principal submatrix.  On output H has been over-
   written by a new Hessenberg matrix that is a perturbation of
   an orthogonal similarity transformation of H.  It is to be
   hoped that the final version of H has many zero subdiagonal
   entries.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine slaqr2(WANTT, WANTZ, N, KTOP, KBOT, NW, H, LDH, ILOZ, IHIZ, Z, LDZ,
                  NS, ND, SR, SI, V, LDV, NH, T, LDT, NV, WV, LDWV, WORK,
                  LWORK)
```

C specification:

```
#include "armpl.h"

void slaqr2_(const armpl_int_t *wantt, const armpl_int_t *wantz,
             const armpl_int_t *n, const armpl_int_t *ktop,
             const armpl_int_t *kbot, const armpl_int_t *nw, float *h,
             const armpl_int_t *ldh, const armpl_int_t *iloz,
             const armpl_int_t *ihiz, float *z, const armpl_int_t *ldz,
             armpl_int_t *ns, armpl_int_t *nd, float *sr, float *si, float *v,
             const armpl_int_t *ldv, const armpl_int_t *nh, float *t,
             const armpl_int_t *ldt, const armpl_int_t *nv, float *wv,
             const armpl_int_t *ldwv, float *work, const armpl_int_t *lwork);
```

Parameters

WANTT Input parameter.

WANTT is LOGICAL

If .TRUE., then the Hessenberg matrix H is fully updated so that the quasi-triangular Schur factor may be computed (in cooperation with the calling subroutine). If .FALSE., then only enough of H is updated to preserve the eigenvalues.

WANTZ Input parameter.

WANTZ is LOGICAL

If .TRUE., then the orthogonal matrix Z is updated so so that the orthogonal Schur factor may be computed (in cooperation with the calling subroutine). If .FALSE., then Z is not referenced.

N Input parameter.

N is INTEGER

The order of the matrix H and (if WANTZ is .TRUE.) the order of the orthogonal matrix Z.

KTOP Input parameter.

KTOP is INTEGER

It is assumed that either KTOP = 1 or H(KTOP,KTOP-1)=0. KBOT and KTOP together determine an isolated block along the diagonal of the Hessenberg matrix.

KBOT Input parameter.

KBOT is INTEGER

It is assumed without a check that either KBOT = N or H(KBOT+1, KBOT)=0. KBOT and KTOP together determine an isolated block along the diagonal of the Hessenberg matrix.

NW Input parameter.

NW is INTEGER

Deflation window size. 1 .LE. NW .LE. (KBOT-KTOP+1).

H Input and output parameter.

H is REAL

H is an array, dimension (LDH, N). On input the initial N-by-N section of H stores the Hessenberg matrix undergoing aggressive early deflation. On output H has been transformed by an orthogonal similarity transformation, perturbed, and the returned to Hessenberg form that (it is to be hoped) has some zero subdiagonal entries.

LDH Input parameter.

LDH is INTEGER

Leading dimension of H just as declared in the calling subroutine. N .LE. LDH

ILOZ Input parameter.

ILOZ is INTEGER

IHIZ Input parameter.

IHIZ is INTEGER

Specify the rows of Z to which transformations must be applied if WANTZ is .TRUE.. 1 .LE. ILOZ .LE. IHIZ .LE. N.

Z Input and output parameter.

Z is REAL

Z is an array, dimension (LDZ, N). IF WANTZ is .TRUE., then on output, the orthogonal similarity transformation mentioned above has been accumulated into Z(ILOZ:IHIZ,ILOZ:IHIZ) from the right. If WANTZ is .FALSE., then Z is unreferenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of Z just as declared in the calling subroutine. 1 .LE. LDZ.

NS Output parameter.

NS is INTEGER

The number of unconverged (ie approximate) eigenvalues returned in SR and SI that may be used as shifts by the calling subroutine.

ND Output parameter.

ND is INTEGER

The number of converged eigenvalues uncovered by this subroutine.

SR Output parameter.

SR is REAL

SR is an array, dimension (KBOT) .

SI Output parameter.

SI is REAL

SI is an array, dimension (KBOT). On output, the real and imaginary parts of approximate eigenvalues that may be used for shifts are stored in SR(KBOT-ND-NS+1) through SR(KBOT-ND) and SI(KBOT-ND-NS+1) through SI(KBOT-ND), respectively. The real and imaginary parts of converged eigenvalues are stored in SR(KBOT-ND+1) through SR(KBOT) and SI(KBOT-ND+1) through SI(KBOT), respectively.

V Output parameter.

V is REAL

V is an array, dimension (LDV, NW). An NW-by-NW work array.

LDV Input parameter.

LDV is INTEGER

The leading dimension of V just as declared in the calling subroutine. NW .LE. LDV

NH Input parameter.

NH is INTEGER

The number of columns of T. NH.GE.NW.

T Output parameter.

T is REAL

T is an array, dimension (LDT, NW) .

LDT Input parameter.

LDT is INTEGER

The leading dimension of T just as declared in the calling subroutine. NW .LE. LDT

NV Input parameter.

NV is INTEGER

The number of rows of work array WV available for workspace. NV.GE.NW.

WV Output parameter.

WV is REAL

WV is an array, dimension (LDWV, NW) .

LDWV Input parameter.

LDWV is INTEGER

The leading dimension of W just as declared in the calling subroutine. NW .LE. LDV

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (LWORK). On exit, WORK(1) is set to an estimate of the optimal value of LWORK for the given values of N, NW, KTOP and KBOT.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the work array WORK. LWORK = 2*NW suffices, but greater efficiency may result from larger values of LWORK.

If LWORK = -1, then a workspace query is assumed; SLAQR2 only estimates the optimal workspace size for the given values of N, NW, KTOP and KBOT. The estimate is returned in WORK(1). No error message related to LWORK is issued by XERBLA. Neither H nor Z are accessed.

Related Information

For this routine in other precisions, please see *claqr2*, *dlaqr2* and *zlaqr2*.

4.17.509 slaqr3

```
Aggressive early deflation:

   SLAQR3 accepts as input an upper Hessenberg matrix
   H and performs an orthogonal similarity transformation
   designed to detect and deflate fully converged eigenvalues from
   a trailing principal submatrix.  On output H has been over-
   written by a new Hessenberg matrix that is a perturbation of
   an orthogonal similarity transformation of H.  It is to be
   hoped that the final version of H has many zero subdiagonal
   entries.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine slaqr3(WANTT, WANTZ, N, KTOP, KBOT, NW, H, LDH, ILOZ, IHIZ, Z, LDZ,
                  NS, ND, SR, SI, V, LDV, NH, T, LDT, NV, WV, LDWV, WORK,
                  LWORK)
```

C specification:

```
#include "armpl.h"

void slaqr3_(const armpl_int_t *wantt, const armpl_int_t *wantz,
             const armpl_int_t *n, const armpl_int_t *ktop,
             const armpl_int_t *kbot, const armpl_int_t *nw, float *h,
             const armpl_int_t *ldh, const armpl_int_t *iloz,
             const armpl_int_t *ihiz, float *z, const armpl_int_t *ldz,
             armpl_int_t *ns, armpl_int_t *nd, float *sr, float *si, float *v,
             const armpl_int_t *ldv, const armpl_int_t *nh, float *t,
             const armpl_int_t *ldt, const armpl_int_t *nv, float *wv,
             const armpl_int_t *ldwv, float *work, const armpl_int_t *lwork);
```

Parameters

WANTT Input parameter.

WANTT is LOGICAL

If .TRUE., then the Hessenberg matrix H is fully updated so that the quasi-triangular Schur factor may be computed (in cooperation with the calling subroutine). If .FALSE., then only enough of H is updated to preserve the eigenvalues.

WANTZ Input parameter.

WANTZ is LOGICAL

If .TRUE., then the orthogonal matrix Z is updated so so that the orthogonal Schur factor may be computed (in cooperation with the calling subroutine). If .FALSE., then Z is not referenced.

N Input parameter.

N is INTEGER

The order of the matrix H and (if WANTZ is .TRUE.) the order of the orthogonal matrix Z.

KTOP Input parameter.

KTOP is INTEGER

It is assumed that either KTOP = 1 or H(KTOP,KTOP-1)=0. KBOT and KTOP together determine an isolated block along the diagonal of the Hessenberg matrix.

KBOT Input parameter.

KBOT is INTEGER

It is assumed without a check that either KBOT = N or H(KBOT+1, KBOT)=0. KBOT and KTOP together determine an isolated block along the diagonal of the Hessenberg matrix.

NW Input parameter.

NW is INTEGER

Deflation window size. 1 .LE. NW .LE. (KBOT-KTOP+1).

H Input and output parameter.

H is REAL

H is an array, dimension (LDH, N). On input the initial N-by-N section of H stores the Hessenberg matrix undergoing aggressive early deflation. On output H has been transformed by an orthogonal similarity transformation, perturbed, and the returned to Hessenberg form that (it is to be hoped) has some zero subdiagonal entries.

LDH Input parameter.

LDH is INTEGER

Leading dimension of H just as declared in the calling subroutine. N .LE. LDH

ILOZ Input parameter.

ILOZ is INTEGER

IHIZ Input parameter.

IHIZ is INTEGER

Specify the rows of Z to which transformations must be applied if WANTZ is .TRUE.. 1 .LE. ILOZ .LE. IHIZ .LE. N.

Z Input and output parameter.

Z is REAL

Z is an array, dimension (LDZ, N). IF WANTZ is .TRUE., then on output, the orthogonal similarity transformation mentioned above has been accumulated into Z(ILOZ:IHIZ,ILOZ:IHIZ) from the right. If WANTZ is .FALSE., then Z is unreferenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of Z just as declared in the calling subroutine. 1 .LE. LDZ.

NS Output parameter.

NS is INTEGER

The number of unconverged (ie approximate) eigenvalues returned in SR and SI that may be used as shifts by the calling subroutine.

ND Output parameter.

ND is INTEGER

The number of converged eigenvalues uncovered by this subroutine.

SR Output parameter.

SR is REAL

SR is an array, dimension (KBOT) .

SI Output parameter.

SI is REAL

SI is an array, dimension (KBOT). On output, the real and imaginary parts of approximate eigenvalues that may be used for shifts are stored in SR(KBOT-ND-NS+1) through SR(KBOT-ND) and SI(KBOT-ND-NS+1) through SI(KBOT-ND), respectively. The real and imaginary parts of converged eigenvalues are stored in SR(KBOT-ND+1) through SR(KBOT) and SI(KBOT-ND+1) through SI(KBOT), respectively.

V Output parameter.

V is REAL

V is an array, dimension (LDV, NW). An NW-by-NW work array.

LDV Input parameter.

LDV is INTEGER

The leading dimension of V just as declared in the calling subroutine. NW .LE. LDV

NH Input parameter.

NH is INTEGER

The number of columns of T. NH.GE.NW.

T Output parameter.

T is REAL

T is an array, dimension (LDT, NW) .

LDT Input parameter.

LDT is INTEGER

The leading dimension of T just as declared in the calling subroutine. NW .LE. LDT

NV Input parameter.

NV is INTEGER

The number of rows of work array WV available for workspace. NV.GE.NW.

WV Output parameter.

WV is REAL

WV is an array, dimension (LDWV, NW) .

LDWV Input parameter.

LDWV is INTEGER

The leading dimension of W just as declared in the calling subroutine. NW .LE. LDV

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (LWORK). On exit, WORK(1) is set to an estimate of the optimal value of LWORK for the given values of N, NW, KTOP and KBOT.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the work array WORK. LWORK = 2*NW suffices, but greater efficiency may result from larger values of LWORK.

If LWORK = -1, then a workspace query is assumed; SLAQR3 only estimates the optimal workspace size for the given values of N, NW, KTOP and KBOT. The estimate is returned in WORK(1). No error message related to LWORK is issued by XERBLA. Neither H nor Z are accessed.

Related Information

For this routine in other precisions, please see *claqr3*, *dlaqr3* and *zlaqr3*.

4.17.510 slaqr4

```
SLAQR4 implements one level of recursion for SLAQR0.
   It is a complete implementation of the small bulge multi-shift
   QR algorithm.  It may be called by SLAQR0 and, for large enough
   deflation window size, it may be called by SLAQR3.  This
   subroutine is identical to SLAQR0 except that it calls SLAQR2
   instead of SLAQR3.
```

(continues on next page)

(continued from previous page)

```
   SLAQR4 computes the eigenvalues of a Hessenberg matrix H
   and, optionally, the matrices T and Z from the Schur decomposition
   H = Z T Z**T, where T is an upper quasi-triangular matrix (the
   Schur form), and Z is the orthogonal matrix of Schur vectors.

   Optionally Z may be postmultiplied into an input orthogonal
   matrix Q so that this routine can give the Schur factorization
   of a matrix A which has been reduced to the Hessenberg form H
   by the orthogonal matrix Q:  A = Q*H*Q**T = (QZ)*T*(QZ)**T.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine slaqr4(WANTT, WANTZ, N, ILO, IHI, H, LDH, WR, WI, ILOZ, IHIZ, Z,
                  LDZ, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void slaqr4_(const armpl_int_t *wantt, const armpl_int_t *wantz,
             const armpl_int_t *n, const armpl_int_t *ilo,
             const armpl_int_t *ihi, float *h, const armpl_int_t *ldh,
             float *wr, float *wi, const armpl_int_t *iloz,
             const armpl_int_t *ihiz, float *z, const armpl_int_t *ldz,
             float *work, const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

WANTT Input parameter.

WANTT is LOGICAL

= .TRUE. : the full Schur form T is required; = .FALSE.: only eigenvalues are required.

WANTZ Input parameter.

WANTZ is LOGICAL

= .TRUE. : the matrix of Schur vectors Z is required; = .FALSE.: Schur vectors are not required.

N Input parameter.

N is INTEGER

The order of the matrix H. N .GE. 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

It is assumed that H is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N and, if ILO.GT.1, H(ILO,ILO-1) is zero. ILO and IHI are normally set by a previous call to SGEBAL, and then passed to SGEHRD when the matrix output by SGEBAL is reduced to Hessenberg form. Otherwise, ILO and IHI

should be set to 1 and N, respectively. If N.GT.0, then 1.LE.ILO.LE.IHI.LE.N. If N = 0, then ILO = 1 and IHI = 0.

H Input and output parameter.

H is REAL

H is an array, dimension (LDH, N). On entry, the upper Hessenberg matrix H. On exit, if INFO = 0 and WANTT is .TRUE., then H contains the upper quasi-triangular matrix T from the Schur decomposition (the Schur form); 2-by-2 diagonal blocks (corresponding to complex conjugate pairs of eigenvalues) are returned in standard form, with H(i,i) = H(i+1,i+1) and H(i+1,i)*H(i,i+1).LT.0. If INFO = 0 and WANTT is .FALSE., then the contents of H are unspecified on exit. (The output value of H when INFO.GT.0 is given under the description of INFO below.)

This subroutine may explicitly set H(i,j) = 0 for i.GT.j and j = 1, 2, ... ILO-1 or j = IHI+1, IHI+2, ... N.

LDH Input parameter.

LDH is INTEGER

The leading dimension of the array H. LDH .GE. max(1, N).

WR Output parameter.

WR is REAL

WR is an array, dimension (IHI) .

WI Output parameter.

WI is REAL

WI is an array, dimension (IHI). The real and imaginary parts, respectively, of the computed eigenvalues of H(ILO:IHI,ILO:IHI) are stored in WR(ILO:IHI) and WI(ILO:IHI). If two eigenvalues are computed as a complex conjugate pair, they are stored in consecutive elements of WR and WI, say the i-th and (i+1)th, with WI(i) .GT. 0 and WI(i+1) .LT. 0. If WANTT is .TRUE., then the eigenvalues are stored in the same order as on the diagonal of the Schur form returned in H, with WR(i) = H(i,i) and, if H(i:i+1,i:i+1) is a 2-by-2 diagonal block, WI(i) = sqrt(-H(i+1,i)*H(i,i+1)) and WI(i+1) = -WI(i).

ILOZ Input parameter.

ILOZ is INTEGER

IHIZ Input parameter.

IHIZ is INTEGER

Specify the rows of Z to which transformations must be applied if WANTZ is .TRUE.. 1 .LE. ILOZ .LE. ILO; IHI .LE. IHIZ .LE. N.

Z Input and output parameter.

Z is REAL

Z is an array, dimension (LDZ, IHI). If WANTZ is .FALSE., then Z is not referenced. If WANTZ is .TRUE., then Z(ILO:IHI,ILOZ:IHIZ) is replaced by Z(ILO:IHI,ILOZ:IHIZ)*U where U is the orthogonal Schur factor of H(ILO:IHI,ILO:IHI). (The output value of Z when INFO.GT.0 is given under the description of INFO below.)

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. if WANTZ is .TRUE. then LDZ.GE.MAX(1, IHIZ). Otherwize, LDZ.GE.1.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension LWORK. On exit, if LWORK = -1, WORK(1) returns an estimate of the optimal value for LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK .GE. max(1, N) is sufficient, but LWORK typically as large as 6* N may be required for optimal performance. A workspace query to determine the optimal workspace size is recommended.

If LWORK = -1, then SLAQR4 does a workspace query. In this case, SLAQR4 checks the input parameters and estimates the optimal workspace size for the given values of N, ILO and IHI. The estimate is returned in WORK(1). No error message related to LWORK is issued by XERBLA. Neither H nor Z are accessed.

INFO Output parameter.

INFO is INTEGER

verbatim INFO is INTEGER = 0: successful exit .GT. 0: if INFO = i, SLAQR4 failed to compute all of the eigenvalues. Elements 1:ilo-1 and i+1:n of WR and WI contain those eigenvalues which have been successfully computed. (Failures are rare.)

If INFO .GT. 0 and WANT is .FALSE., then on exit, the remaining unconverged eigenvalues are the eigenvalues of the upper Hessenberg matrix rows and columns ILO through INFO of the final, output value of H.

If INFO .GT. 0 and WANTT is .TRUE., then on exit

(*) (initial value of H)*U = U*(final value of H)

where U is a orthogonal matrix. The final value of H is upper Hessenberg and triangular in rows and columns INFO+1 through IHI.

If INFO .GT. 0 and WANTZ is .TRUE., then on exit

(final value of Z(ILO:IHI,ILOZ:IHIZ) = (initial value of Z(ILO:IHI,ILOZ:IHIZ)*U

where U is the orthogonal matrix in (*) (regard- less of the value of WANTT.)

If INFO .GT. 0 and WANTZ is .FALSE., then Z is not accessed.

Related Information

For this routine in other precisions, please see *claqr4*, *dlaqr4* and *zlaqr4*.

4.17.511 slaqr5

```
SLAQR5, called by SLAQR0, performs a
   single small-bulge multi-shift QR sweep.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine slaqr5(WANTT, WANTZ, KACC22, N, KTOP, KBOT, NSHFTS, SR, SI, H, LDH,
                  ILOZ, IHIZ, Z, LDZ, V, LDV, U, LDU, NV, WV, LDWV, NH, WH,
                  LDWH)
```

C specification:

```
#include "armpl.h"

void slaqr5_(const armpl_int_t *wantt, const armpl_int_t *wantz,
```

(continues on next page)

(continued from previous page)

```
                const armpl_int_t *kacc22, const armpl_int_t *n,
                const armpl_int_t *ktop, const armpl_int_t *kbot,
                const armpl_int_t *nshfts, float *sr, float *si, float *h,
                const armpl_int_t *ldh, const armpl_int_t *iloz,
                const armpl_int_t *ihiz, float *z, const armpl_int_t *ldz,
                float *v, const armpl_int_t *ldv, float *u,
                const armpl_int_t *ldu, const armpl_int_t *nv, float *wv,
                const armpl_int_t *ldwv, const armpl_int_t *nh, float *wh,
                const armpl_int_t *ldwh);
```

Parameters

WANTT Input parameter.

WANTT is LOGICAL

WANTT = .true. if the quasi-triangular Schur factor is being computed. WANTT is set to .false. otherwise.

WANTZ Input parameter.

WANTZ is LOGICAL

WANTZ = .true. if the orthogonal Schur factor is being computed. WANTZ is set to .false. otherwise.

KACC22 Input parameter.

KACC22 is INTEGER with value 0, 1, or 2.

Specifies the computation mode of far-from-diagonal orthogonal updates. = 0: SLAQR5 does not accumulate reflections and does not use matrix-matrix multiply to update far-from-diagonal matrix entries. = 1: SLAQR5 accumulates reflections and uses matrix-matrix multiply to update the far-from-diagonal matrix entries. = 2: SLAQR5 accumulates reflections, uses matrix-matrix multiply to update the far-from-diagonal matrix entries, and takes advantage of 2-by-2 block structure during matrix multiplies.

N Input parameter.

N is INTEGER

N is the order of the Hessenberg matrix H upon which this subroutine operates.

KTOP Input parameter.

KTOP is INTEGER

KBOT Input parameter.

KBOT is INTEGER

These are the first and last rows and columns of an isolated diagonal block upon which the QR sweep is to be applied. It is assumed without a check that either KTOP = 1 or H(KTOP,KTOP-1) = 0 and either KBOT = N or H(KBOT+1, KBOT) = 0.

NSHFTS Input parameter.

NSHFTS is INTEGER

NSHFTS gives the number of simultaneous shifts. NSHFTS must be positive and even.

SR Input and output parameter.

SR is REAL

SR is an array, dimension (NSHFTS) .

SI Input and output parameter.

SI is REAL

SI is an array, dimension (NSHFTS). SR contains the real parts and SI contains the imaginary parts of the NSHFTS shifts of origin that define the multi-shift QR sweep. On output SR and SI may be reordered.

H Input and output parameter.

H is REAL

H is an array, dimension (LDH, N). On input H contains a Hessenberg matrix. On output a multi-shift QR sweep with shifts SR(J)+i*SI(J) is applied to the isolated diagonal block in rows and columns KTOP through KBOT.

LDH Input parameter.

LDH is INTEGER

LDH is the leading dimension of H just as declared in the calling procedure. LDH.GE.MAX(1, N).

ILOZ Input parameter.

ILOZ is INTEGER

IHIZ Input parameter.

IHIZ is INTEGER

Specify the rows of Z to which transformations must be applied if WANTZ is .TRUE.. 1 .LE. ILOZ .LE. IHIZ .LE. N

Z Input and output parameter.

Z is REAL

Z is an array, dimension (LDZ, IHIZ). If WANTZ = .TRUE., then the QR Sweep orthogonal similarity transformation is accumulated into Z(ILOZ:IHIZ,ILOZ:IHIZ) from the right. If WANTZ = .FALSE., then Z is unreferenced.

LDZ Input parameter.

LDZ is INTEGER

LDA is the leading dimension of Z just as declared in the calling procedure. LDZ.GE.N.

V Output parameter.

V is REAL

V is an array, dimension (LDV,NSHFTS/2) .

LDV Input parameter.

LDV is INTEGER

LDV is the leading dimension of V as declared in the calling procedure. LDV.GE.3.

U Output parameter.

U is REAL

U is an array, dimension (LDU,3*NSHFTS-3) .

LDU Input parameter.

LDU is INTEGER

LDU is the leading dimension of U just as declared in the in the calling subroutine. LDU.GE.3*NSHFTS-3.

NH Input parameter.

NH is INTEGER

NH is the number of columns in array WH available for workspace. NH.GE.1.

WH Output parameter.

WH is REAL

WH is an array, dimension (LDWH, NH) .

LDWH Input parameter.

LDWH is INTEGER

Leading dimension of WH just as declared in the calling procedure. LDWH.GE.3*NSHFTS-3.

NV Input parameter.

NV is INTEGER

NV is the number of rows in WV agailable for workspace. NV.GE.1.

WV Output parameter.

WV is REAL

WV is an array, dimension (LDWV,3*NSHFTS-3) .

LDWV Input parameter.

LDWV is INTEGER

LDWV is the leading dimension of WV as declared in the in the calling subroutine. LDWV.GE.NV.

Related Information

For this routine in other precisions, please see *claqr5*, *dlaqr5* and *zlaqr5*.

4.17.512 slaqsb

slaqsb equilibrates a symmetric band matrix A using the scaling factors in the vector S.

Syntax

Fortran specification:

```
use armpl_library

subroutine slaqsb(UPLO, N, KD, AB, LDAB, S, SCOND, AMAX, EQUED)
```

C specification:

```
#include "armpl.h"

void slaqsb_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
             float *ab, const armpl_int_t *ldab, const float *s,
             const float *scond, const float *amax, char *equed, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored. = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of super-diagonals of the matrix A if UPLO = 'U', or the number of sub-diagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is REAL

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, if INFO = 0, the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T of the band matrix A, in the same storage format as A.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

S Input parameter.

S is REAL

S is an array, dimension (N). The scale factors for A.

SCOND Input parameter.

SCOND is REAL

Ratio of the smallest S(i) to the largest S(i).

AMAX Input parameter.

AMAX is REAL

Absolute value of largest matrix entry.

EQUED Output parameter.

EQUED is CHARACTER*1

Specifies whether or not equilibration was done. = 'N': No equilibration. = 'Y': Equilibration was done, i.e., A has been replaced by diag(S) * A * diag(S).

Related Information

For this routine in other precisions, please see *claqsb*, *dlaqsb* and *zlaqsb*.

4.17.513 slaqsp

slaqsp equilibrates a symmetric matrix A using the scaling factors in the vector S.

Syntax

Fortran specification:

```
use armpl_library

subroutine slaqsp(UPLO, N, AP, S, SCOND, AMAX, EQUED)
```

C specification:

```
#include "armpl.h"

void slaqsp_(const char *uplo, const armpl_int_t *n, float *ap,
             const float *s, const float *scond, const float *amax,
             char *equed, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored. = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is REAL

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

On exit, the equilibrated matrix: diag(S) * A * diag(S), in the same storage format as A.

S Input parameter.

S is REAL

S is an array, dimension (N). The scale factors for A.

SCOND Input parameter.

SCOND is REAL

Ratio of the smallest S(i) to the largest S(i).

AMAX Input parameter.

AMAX is REAL

Absolute value of largest matrix entry.

EQUED Output parameter.

EQUED is CHARACTER*1

Specifies whether or not equilibration was done. = 'N': No equilibration. = 'Y': Equilibration was done, i.e., A has been replaced by diag(S) * A * diag(S).

Related Information

For this routine in other precisions, please see *claqsp*, *dlaqsp* and *zlaqsp*.

4.17.514 slaqsy

slaqsy equilibrates a symmetric matrix A using the scaling factors in the vector S.

Syntax

Fortran specification:

```
use armpl_library

subroutine slaqsy(UPLO, N, A, LDA, S, SCOND, AMAX, EQUED)
```

C specification:

```
#include "armpl.h"

void slaqsy_(const char *uplo, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, const float *s, const float *scond,
             const float *amax, char *equed, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored. = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading n by n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n by n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if EQUED = 'Y', the equilibrated matrix: diag(S) * A * diag(S).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(N,1).

S Input parameter.

S is REAL

S is an array, dimension (N). The scale factors for A.

SCOND Input parameter.

SCOND is REAL

Ratio of the smallest S(i) to the largest S(i).

AMAX Input parameter.

AMAX is REAL

Absolute value of largest matrix entry.

EQUED Output parameter.

EQUED is CHARACTER*1

Specifies whether or not equilibration was done. = 'N': No equilibration. = 'Y': Equilibration was done, i.e., A has been replaced by diag(S) * A * diag(S).

Related Information

For this routine in other precisions, please see *claqsy*, *dlaqsy* and *zlaqsy*.

4.17.515 slaqtr

slaqtr solves the real quasi-triangular system

```
op(T)*p = scale*c,                 if LREAL = .TRUE.
```

or the complex quasi-triangular systems

```
op(T + iB)*(p+iq) = scale*(c+id),  if LREAL = .FALSE.
```

in real arithmetic, where T is upper quasi-triangular. If LREAL = .FALSE., then the first diagonal block of T must be 1 by 1, B is the specially structured matrix

```
B = [ b(1) b(2) ... b(n) ]
    [       w            ]
    [           w        ]
    [               .    ]
    [                   w ]
```

op(A) = A or A^T , A^T denotes the transpose of matrix A.

On input, X = [c]. On output, X = [p].

```
[ d ]                  [ q ]
```

This subroutine is designed for the condition number estimation in routine STRSNA.

Syntax

Fortran specification:

```
use armpl_library

subroutine slaqtr(LTRAN, LREAL, N, T, LDT, B, W, SCALE, X, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void slaqtr_(const armpl_int_t *ltran, const armpl_int_t *lreal,
             const armpl_int_t *n, const float *t, const armpl_int_t *ldt,
             const float *b, const float *w, float *scale, float *x,
             float *work, armpl_int_t *info);
```

Parameters

LTRAN Input parameter.

LTRAN is LOGICAL

On entry, LTRAN specifies the option of conjugate transpose: = .FALSE., op(T+i*B) = T+i*B, = .TRUE., op(T+i*B) = $(T+i*B)^T$.

LREAL Input parameter.

L‘‘REAL‘‘ is LOGICAL

On entry, L‘‘REAL‘‘ specifies the input matrix structure: = .FALSE., the input is complex = .TRUE., the input is real

N Input parameter.

N is INTEGER

On entry, N specifies the order of T+i*B. N >= 0.

T Input parameter.

T is REAL

T is an array, dimension (LDT, N). On entry, T contains a matrix in Schur canonical form. If L‘‘REAL‘‘ = .FALSE., then the first diagonal block of T must be 1 by 1.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the matrix T. LDT >= max(1, N).

B Input parameter.

B is REAL

B is an array, dimension (N). On entry, B contains the elements to form the matrix B as described above. If L‘‘REAL‘‘ = .TRUE., B is not referenced.

W Input parameter.

W is REAL

On entry, W is the diagonal element of the matrix B. If L‘‘REAL‘‘ = .TRUE., W is not referenced.

SCALE Output parameter.

SCALE is REAL

On exit, SCALE is the scale factor.

X Input and output parameter.

X is REAL

X is an array, dimension (2*N). On entry, X contains the right hand side of the system. On exit, X is overwritten by the solution.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

On exit, INFO is set to 0: successful exit. 1: the some diagonal 1 by 1 block has been perturbed by a small number SMIN to keep nonsingularity. 2: the some diagonal 2 by 2 block has been perturbed by a small

number in SLALN2 to keep nonsingularity. NOTE: In the interests of speed, this routine does not check the inputs for errors.

Related Information

For this routine in other precisions, please see *dlaqtr*.

4.17.516 slar1v

`slar1v` computes the (scaled) r-th column of the inverse of the sumbmatrix in rows B1 through BN of the tridiagonal matrix L D L^T - sigma I. When sigma is close to an eigenvalue, the computed vector is an accurate eigenvector. Usually, r corresponds to the index where the eigenvector is largest in magnitude. The following steps accomplish this computation : (a) Stationary qd transform, L D L^T - sigma I = L(+) D(+) $L(+)^T$, (b) Progressive qd transform, L D L^T - sigma I = U(-) D(-) $U(-)^T$, (c) Computation of the diagonal elements of the inverse of

```
L D L**T - sigma I by combining the above transforms, and choosing
r as the index where the diagonal of the inverse is (one of the)
largest in magnitude.
```

(d) Computation of the (scaled) r-th column of the inverse using the

```
twisted factorization obtained by combining the top part of the
the stationary and the bottom part of the progressive transform.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine slar1v(N, B1, BN, LAMBDA, D, L, LD, LLD, PIVMIN, GAPTOL, Z, WANTNC,
                  NEGCNT, ZTZ, MINGMA, R, ISUPPZ, NRMINV, RESID, RQCORR,
                  WORK)
```

C specification:

```
#include "armpl.h"

void slar1v_(const armpl_int_t *n, const armpl_int_t *b1,
             const armpl_int_t *bn, const float *lambda, const float *d,
             const float *l, const float *ld, const float *lld,
             const float *pivmin, const float *gaptol, float *z,
             const armpl_int_t *wantnc, armpl_int_t *negcnt, float *ztz,
             float *mingma, armpl_int_t *r, armpl_int_t *isuppz,
             float *nrminv, float *resid, float *rqcorr, float *work);
```

Parameters

N Input parameter.

`N` is `INTEGER`

The order of the matrix `L D` L^T .

B1 Input parameter.

`B1` is `INTEGER`

First index of the submatrix of `L D` L^T .

BN Input parameter.

BN is INTEGER

Last index of the submatrix of L D L^T .

LAMBDA Input parameter.

LAMBDA is REAL

The shift. In order to compute an accurate eigenvector, LAMBDA should be a good approximation to an eigenvalue of L D L^T .

L Input parameter.

L is REAL

L is an array, dimension (N-1). The (n-1) subdiagonal elements of the unit bidiagonal matrix L, in elements 1 to N-1.

D Input parameter.

D is REAL

D is an array, dimension (N). The n diagonal elements of the diagonal matrix D.

LD Input parameter.

LD is REAL

LD is an array, dimension (N-1). The n-1 elements L(i)*D(i).

LLD Input parameter.

LLD is REAL

LLD is an array, dimension (N-1). The n-1 elements L(i)*L(i)*D(i).

PIVMIN Input parameter.

PIVMIN is REAL

The minimum pivot in the Sturm sequence.

GAPTOL Input parameter.

GAPTOL is REAL

Tolerance that indicates when eigenvector entries are negligible w.r.t. their contribution to the residual.

Z Input and output parameter.

Z is REAL

Z is an array, dimension (N). On input, all entries of Z must be set to 0. On output, Z contains the (scaled) r-th column of the inverse. The scaling is such that Z(R) equals 1.

WANTNC Input parameter.

WANTNC is LOGICAL

Specifies whether NEGCNT has to be computed.

NEGCNT Output parameter.

NEGCNT is INTEGER

If WANTNC is .TRUE. then NEGCNT = the number of pivots < pivmin in the matrix factorization L D L^T , and NEGCNT = -1 otherwise.

ZTZ Output parameter.

ZTZ is REAL

The square of the 2-norm of Z.

MINGMA Output parameter.

MINGMA is REAL

The reciprocal of the largest (in magnitude) diagonal element of the inverse of L D L^T - sigma I.

R Input and output parameter.

R is INTEGER

The twist index for the twisted factorization used to compute Z. On input, 0 <= R <= N. If R is input as 0, R is set to the index where (L D L^T - sigma I)^{-1} is largest in magnitude. If 1 <= R <= N, R is unchanged. On output, R contains the twist index used to compute Z. Ideally, R designates the position of the maximum entry in the eigenvector.

ISUPPZ Output parameter.

ISUPPZ is INTEGER array, dimension (2)

The support of the vector in Z, i.e., the vector Z is nonzero only in elements ISUPPZ(1) through ISUPPZ(2).

NRMINV Output parameter.

NRMINV is REAL

NRMINV = 1/SQRT(ZTZ)

RESID Output parameter.

RESID is REAL

The residual of the FP vector. RESID = ABS(MINGMA)/SQRT(ZTZ)

RQCORR Output parameter.

RQCORR is REAL

The Rayleigh Quotient correction to LAMBDA. RQCORR = MINGMA*TMP

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (4*N) .

Related Information

For this routine in other precisions, please see *clar1v*, *dlar1v* and *zlar1v*.

4.17.517 slar2v

slar2v applies a vector of real plane rotations from both sides to a sequence of 2-by-2 real symmetric matrices, defined by the elements of the vectors x, y and z. For i = 1,2,...,n

```
( x(i)  z(i) ) := (  c(i)  s(i) ) ( x(i)  z(i) ) ( c(i) -s(i) )
( z(i)  y(i) )    ( -s(i)  c(i) ) ( z(i)  y(i) ) ( s(i)  c(i) )
```

Syntax

Fortran specification:

```
use armpl_library

subroutine slar2v(N, X, Y, Z, INCX, C, S, INCC)
```

C specification:

```
#include "armpl.h"

void slar2v_(const armpl_int_t *n, float *x, float *y, float *z,
             const armpl_int_t *incx, const float *c, const float *s,
             const armpl_int_t *incc);
```

Parameters

N Input parameter.

N is INTEGER

The number of plane rotations to be applied.

X Input and output parameter.

X is REAL

X is an array,. dimension (1+(N-1)*INCX) The vector x.

Y Input and output parameter.

Y is REAL

Y is an array,. dimension (1+(N-1)*INCX) The vector y.

Z Input and output parameter.

Z is REAL

Z is an array,. dimension (1+(N-1)*INCX) The vector z.

INCX Input parameter.

INCX is INTEGER

The increment between elements of X, Y and Z. INCX > 0.

C Input parameter.

C is REAL

C is an array, dimension (1+(N-1)*INCC). The cosines of the plane rotations.

S Input parameter.

S is REAL

S is an array, dimension (1+(N-1)*INCC). The sines of the plane rotations.

INCC Input parameter.

INCC is INTEGER

The increment between elements of C and S. INCC > 0.

Related Information

For this routine in other precisions, please see *clar2v*, *dlar2v* and *zlar2v*.

4.17.518 slarf

slarf applies a real elementary reflector H to a real m by n matrix C, from either the left or the right. H is represented in the form

```
H = I - tau * v * v**T
```

where tau is a real scalar and v is a real vector.

If tau = 0, then H is taken to be the unit matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine slarf(SIDE, M, N, V, INCV, TAU, C, LDC, WORK)
```

C specification:

```
#include "armpl.h"

void slarf_(const char *side, const armpl_int_t *m, const armpl_int_t *n,
            const float *v, const armpl_int_t *incv, const float *tau,
            float *c, const armpl_int_t *ldc, float *work, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': form H * C = 'R': form C * H

M Input parameter.

M is INTEGER

The number of rows of the matrix C.

N Input parameter.

N is INTEGER

The number of columns of the matrix C.

V Input parameter.

V is REAL

V is an array, dimension. (1 + (M-1)*abs(INCV)) if SIDE = 'L' or (1 + (N-1)*abs(INCV)) if SIDE = 'R' The vector v in the representation of H. V is not used if TAU = 0.

INCV Input parameter.

INCV is INTEGER

The increment between elements of v. INCV <> 0.

TAU Input parameter.

TAU is REAL

The value tau in the representation of H.

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, N). On entry, the m by n matrix C. On exit, C is overwritten by the matrix H * C if SIDE = 'L', or C * H if SIDE = 'R'.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension. (N) if SIDE = 'L' or (M) if SIDE = 'R'

Related Information

For this routine in other precisions, please see *clarf*, *dlarf* and *zlarf*.

4.17.519 slarfb

slarfb applies a real block reflector H or its transpose H^T to a real m by n matrix C, from either the left or the right.

Syntax

Fortran specification:

```
use armpl_library

subroutine slarfb(SIDE, TRANS, DIRECT, STOREV, M, N, K, V, LDV, T, LDT, C,
                  LDC, WORK, LDWORK)
```

C specification:

```
#include "armpl.h"

void slarfb_(const char *side, const char *trans, const char *direct,
             const char *storev, const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *k, const float *v, const armpl_int_t *ldv,
             const float *t, const armpl_int_t *ldt, float *c,
             const armpl_int_t *ldc, float *work, const armpl_int_t *ldwork,
             ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply H or H^T from the Left = 'R': apply H or H^T from the Right

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply H (No transpose) = 'T': apply H^T (Transpose)

DIRECT Input parameter.

DIRECT is CHARACTER*1

Indicates how H is formed from a product of elementary reflectors = 'F': H = H(1) H(2) . . . H(k) (Forward) = 'B': H = H(k) . . . H(2) H(1) (Backward)

STOREV Input parameter.

STOREV is CHARACTER*1

Indicates how the vectors which define the elementary reflectors are stored: = 'C': Columnwise = 'R': Rowwise

M Input parameter.

M is INTEGER

The number of rows of the matrix C.

N Input parameter.

N is INTEGER

The number of columns of the matrix C.

K Input parameter.

K is INTEGER

The order of the matrix T (= the number of elementary reflectors whose product defines the block reflector).

V Input parameter.

V is REAL

V is an array, dimension. (LDV, K) if STOREV = 'C' (LDV, M) if STOREV = 'R' and SIDE = 'L' (LDV, N) if STOREV = 'R' and SIDE = 'R' The matrix V. See Further Details.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. If STOREV = 'C' and SIDE = 'L', LDV >= max(1, M); if STOREV = 'C' and SIDE = 'R', LDV >= max(1, N); if STOREV = 'R', LDV >= K.

T Input parameter.

T is REAL

T is an array, dimension (LDT, K). The triangular k by k matrix T in the representation of the block reflector.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= K.

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, N). On entry, the m by n matrix C. On exit, C is overwritten by H* C or H^T * C or C*H or C*H^T .

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (LDWORK, K) .

LDWORK Input parameter.

LDWORK is INTEGER

The leading dimension of the array WORK. If SIDE = 'L', LDWORK >= max(1, N); if SIDE = 'R', LDWORK >= max(1, M).

Related Information

For this routine in other precisions, please see *clarfb*, *dlarfb* and *zlarfb*. It also exists with a native C interface as *LAPACKE_slarfb*.

4.17.520 slarfg

`slarfg` generates a real elementary reflector H of order n, such that

```
H * ( alpha ) = ( beta ),   H**T * H = I.
    (   x   )   (   0  )
```

where alpha and beta are scalars, and x is an (n-1)-element real vector. H is represented in the form

```
H = I - tau * ( 1 ) * ( 1 v**T ) ,
              ( v )
```

where tau is a real scalar and v is a real (n-1)-element vector.

If the elements of x are all zero, then tau = 0 and H is taken to be the unit matrix.

Otherwise 1 <= tau <= 2.

Syntax

Fortran specification:

```
use armpl_library

subroutine slarfg(N, ALPHA, X, INCX, TAU)
```

C specification:

```
#include "armpl.h"

void slarfg_(const armpl_int_t *n, float *alpha, float *x,
             const armpl_int_t *incx, float *tau);
```

Parameters

N Input parameter.

N is INTEGER

The order of the elementary reflector.

ALPHA Input and output parameter.

ALPHA is REAL

On entry, the value alpha. On exit, it is overwritten with the value beta.

X Input and output parameter.

X is REAL

X is an array, dimension. (1+(N-2)*abs(INCX)) On entry, the vector x. On exit, it is overwritten with the vector v.

INCX Input parameter.

INCX is INTEGER

The increment between elements of X. INCX > 0.

TAU Output parameter.

TAU is REAL

The value tau.

Related Information

For this routine in other precisions, please see *clarfg*, *dlarfg* and *zlarfg*. It also exists with a native C interface as *LAPACKE_slarfg*.

4.17.521 slarfgp

slarfgp generates a real elementary reflector H of order n, such that

```
H * ( alpha ) = ( beta ),   H**T * H = I.
    (   x   )   (   0  )
```

where alpha and beta are scalars, beta is non-negative, and x is an (n-1)-element real vector. H is represented in the form

```
H = I - tau * ( 1 ) * ( 1 v**T ) ,
              ( v )
```

where tau is a real scalar and v is a real (n-1)-element vector.

If the elements of x are all zero, then tau = 0 and H is taken to be the unit matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine slarfgp(N, ALPHA, X, INCX, TAU)
```

C specification:

```
#include "armpl.h"

void slarfgp_(const armpl_int_t *n, float *alpha, float *x,
              const armpl_int_t *incx, float *tau);
```

Parameters

N Input parameter.

N is INTEGER

The order of the elementary reflector.

ALPHA Input and output parameter.

ALPHA is REAL

On entry, the value alpha. On exit, it is overwritten with the value beta.

X Input and output parameter.

X is REAL

X is an array, dimension. (1+(N-2)*abs(INCX)) On entry, the vector x. On exit, it is overwritten with the vector v.

INCX Input parameter.

INCX is INTEGER

The increment between elements of X. INCX > 0.

TAU Output parameter.

TAU is REAL

The value tau.

Related Information

For this routine in other precisions, please see *clarfgp*, *dlarfgp* and *zlarfgp*.

4.17.522 slarft

slarft forms the triangular factor T of a real block reflector H of order n, which is defined as a product of k elementary reflectors.

If DIRECT = 'F', H = H(1) H(2) . . . H(k) and T is upper triangular;

If DIRECT = 'B', H = H(k) . . . H(2) H(1) and T is lower triangular.

If STOREV = 'C', the vector which defines the elementary reflector H(i) is stored in the i-th column of the array V, and

```
H  =  I - V * T * V**T
```

If STOREV = 'R', the vector which defines the elementary reflector H(i) is stored in the i-th row of the array V, and

```
H  =  I - V**T * T * V
```

Syntax

Fortran specification:

```
use armpl_library

subroutine slarft(DIRECT, STOREV, N, K, V, LDV, TAU, T, LDT)
```

C specification:

```
#include "armpl.h"

void slarft_(const char *direct, const char *storev, const armpl_int_t *n,
             const armpl_int_t *k, const float *v, const armpl_int_t *ldv,
             const float *tau, float *t, const armpl_int_t *ldt, ... );
```

Parameters

DIRECT Input parameter.

DIRECT is CHARACTER*1

Specifies the order in which the elementary reflectors are multiplied to form the block reflector: = 'F': H = H(1) H(2) . . . H(k) (Forward) = 'B': H = H(k) . . . H(2) H(1) (Backward)

STOREV Input parameter.

STOREV is CHARACTER*1

Specifies how the vectors which define the elementary reflectors are stored (see also Further Details): = 'C': columnwise = 'R': rowwise

N Input parameter.

N is INTEGER

The order of the block reflector H. N >= 0.

K Input parameter.

K is INTEGER

The order of the triangular factor T (= the number of elementary reflectors). K >= 1.

V Input parameter.

V is REAL

V is an array, dimension. (LDV, K) if STOREV = 'C' (LDV, N) if STOREV = 'R' The matrix V. See further details.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. If STOREV = 'C', LDV >= max(1, N); if STOREV = 'R', LDV >= K.

TAU Input parameter.

TAU is REAL

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i).

T Output parameter.

T is REAL

T is an array, dimension (LDT, K). The k by k triangular factor T of the block reflector. If DIRECT = 'F', T is upper triangular; if DIRECT = 'B', T is lower triangular. The rest of the array is not used.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= K.

Related Information

For this routine in other precisions, please see *clarft*, *dlarft* and *zlarft*. It also exists with a native C interface as *LAPACKE_slarft*.

4.17.523 slarfx

slarfx applies a real elementary reflector H to a real m by n matrix C, from either the left or the right. H is represented in the form

```
H = I - tau * v * v**T
```

where tau is a real scalar and v is a real vector.

If tau = 0, then H is taken to be the unit matrix

This version uses inline code if H has order < 11.

Syntax

Fortran specification:

```
use armpl_library

subroutine slarfx(SIDE, M, N, V, TAU, C, LDC, WORK)
```

C specification:

```
#include "armpl.h"

void slarfx_(const char *side, const armpl_int_t *m, const armpl_int_t *n,
             const float *v, const float *tau, float *c,
             const armpl_int_t *ldc, float *work, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': form H * C = 'R': form C * H

M Input parameter.

M is INTEGER

The number of rows of the matrix C.

N Input parameter.

N is INTEGER

The number of columns of the matrix C.

V Input parameter.

V is REAL

V is an array, dimension (M) if SIDE = 'L'. or (N) if SIDE = 'R' The vector v in the representation of H.

TAU Input parameter.

TAU is REAL

The value tau in the representation of H.

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, N). On entry, the m by n matrix C. On exit, C is overwritten by the matrix H * C if SIDE = 'L', or C * H if SIDE = 'R'.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDA >= (1, M).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension. (N) if SIDE = 'L' or (M) if SIDE = 'R' WORK is not referenced if H has order < 11.

Related Information

For this routine in other precisions, please see *clarfx*, *dlarfx* and *zlarfx*. It also exists with a native C interface as *LAPACKE_slarfx*.

4.17.524 slarfy

`slarfy` applies an elementary reflector, or Householder matrix, H, to an n x n symmetric matrix C, from both the left and the right.

H is represented in the form

```
H = I - tau * v * v'
```

where tau is a scalar and v is a vector.

If tau is zero, then H is taken to be the unit matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine slarfy(UPLO, N, V, INCV, TAU, C, LDC, WORK)
```

C specification:

```
#include "armpl.h"

void slarfy_(const char *uplo, const armpl_int_t *n, const float *v,
             const armpl_int_t *incv, const float *tau, float *c,
             const armpl_int_t *ldc, float *work, ... );
```

Parameters

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

Specifies whether the upper or lower triangular part of the symmetric matrix `C` is stored. = '`U`': Upper triangle = '`L`': Lower triangle

N Input parameter.

`N` is `INTEGER`

The number of rows and columns of the matrix `C`. `N` >= 0.

V Input parameter.

`V` is `REAL`

`V` is an array, dimension. (1 + (N-1)*abs(`INCV`)) The vector v as described above.

INCV Input parameter.

`INCV` is `INTEGER`

The increment between successive elements of v. `INCV` must not be zero.

TAU Input parameter.

TAU is REAL

The value tau as described above.

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, N). On entry, the matrix C. On exit, C is overwritten by H * C * H'.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (N) .

Related Information

For this routine in other precisions, please see *clarfy*, *dlarfy* and *zlarfy*.

4.17.525 slargv

slargv generates a vector of real plane rotations, determined by elements of the real vectors x and y. For i = 1,2,...,n

```
(  c(i)  s(i) ) ( x(i) ) = ( a(i) )
( -s(i)  c(i) ) ( y(i) ) = (   0  )
```

Syntax

Fortran specification:

```
use armpl_library

subroutine slargv(N, X, INCX, Y, INCY, C, INCC)
```

C specification:

```
#include "armpl.h"

void slargv_(const armpl_int_t *n, float *x, const armpl_int_t *incx,
             float *y, const armpl_int_t *incy, float *c,
             const armpl_int_t *incc);
```

Parameters

N Input parameter.

N is INTEGER

The number of plane rotations to be generated.

X Input and output parameter.

X is REAL

X is an array,. dimension (1+(N-1)*INCX) On entry, the vector x. On exit, x(i) is overwritten by a(i), for i = 1,. . . ,n.

INCX Input parameter.

INCX is INTEGER

The increment between elements of X. INCX > 0.

Y Input and output parameter.

Y is REAL

Y is an array,. dimension (1+(N-1)*INCY) On entry, the vector y. On exit, the sines of the plane rotations.

INCY Input parameter.

INCY is INTEGER

The increment between elements of Y. INCY > 0.

C Output parameter.

C is REAL

C is an array, dimension (1+(N-1)*INCC). The cosines of the plane rotations.

INCC Input parameter.

INCC is INTEGER

The increment between elements of C. INCC > 0.

Related Information

For this routine in other precisions, please see *clargv*, *dlargv* and *zlargv*.

4.17.526 slarnv

slarnv returns a vector of n random real numbers from a uniform or normal distribution.

Syntax

Fortran specification:

```
use armpl_library

subroutine slarnv(IDIST, ISEED, N, X)
```

C specification:

```
#include "armpl.h"

void slarnv_(const armpl_int_t *idist, armpl_int_t *iseed,
             const armpl_int_t *n, float *x);
```

Parameters

IDIST Input parameter.

IDIST is INTEGER

Specifies the distribution of the random numbers: = 1: uniform (0,1) = 2: uniform (-1,1) = 3: normal (0,1)

ISEED Input and output parameter.

ISEED is INTEGER array, dimension (4)

On entry, the seed of the random number generator; the array elements must be between 0 and 4095, and ISEED(4) must be odd. On exit, the seed is updated.

N Input parameter.

N is INTEGER

The number of random numbers to be generated.

X Output parameter.

X is REAL

X is an array, dimension (N). The generated random numbers.

Related Information

For this routine in other precisions, please see *clarnv*, *dlarnv* and *zlarnv*. It also exists with a native C interface as *LAPACKE_slarnv*.

4.17.527 slarra

Compute the splitting points with threshold SPLTOL. slarra sets any "small" off-diagonal elements to zero.

Syntax

Fortran specification:

```
use armpl_library

subroutine slarra(N, D, E, E2, SPLTOL, TNRM, NSPLIT, ISPLIT, INFO)
```

C specification:

```
#include "armpl.h"

void slarra_(const armpl_int_t *n, const float *d, float *e, float *e2,
             const float *spltol, const float *tnrm, armpl_int_t *nsplit,
             armpl_int_t *isplit, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix. N > 0.

D Input parameter.

D is REAL

D is an array, dimension (N). On entry, the N diagonal elements of the tridiagonal matrix T.

E Input and output parameter.

E is REAL

E is an array, dimension (N). On entry, the first (N-1) entries contain the subdiagonal elements of the tridiagonal matrix T; E(N) need not be set. On exit, the entries E(ISPLIT(I)), 1 <= I <= NSPLIT, are set to zero, the other entries of E are untouched.

E2 Input and output parameter.

E2 is REAL

E2 is an array, dimension (N). On entry, the first (N-1) entries contain the SQUARES of the subdiagonal elements of the tridiagonal matrix T; E2(N) need not be set. On exit, the entries E2(ISPLIT(I)), 1 <= I <= NSPLIT, have been set to zero

SPLTOL Input parameter.

SPLTOL is REAL

The threshold for splitting. Two criteria can be used: SPLTOL<0 : criterion based on absolute off-diagonal value SPLTOL>0 : criterion that preserves relative accuracy

TNRM Input parameter.

TNRM is REAL

The norm of the matrix.

NSPLIT Output parameter.

NSPLIT is INTEGER

The number of blocks T splits into. 1 <= NSPLIT <= N.

ISPLIT Output parameter.

ISPLIT is INTEGER array, dimension (N)

The splitting points, at which T breaks up into blocks. The first block consists of rows/columns 1 to ISPLIT(1), the second of rows/columns ISPLIT(1)+1 through ISPLIT(2), etc., and the NSPLIT-th consists of rows/columns ISPLIT(NSPLIT-1)+1 through ISPLIT(NSPLIT)=N.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit

Related Information

For this routine in other precisions, please see *dlarra*.

4.17.528 slarrb

Given the relatively robust representation(RRR) L D L^T, slarrb does "limited" bisection to refine the eigenvalues of L D L^T, W(IFIRST-OFFSET) through W(ILAST-OFFSET), to more accuracy. Initial guesses for these eigenvalues are input in W, the corresponding estimate of the error in these guesses and their gaps are input in WERR and WGAP, respectively. During bisection, intervals [left, right] are maintained by storing their mid-points and semi-widths in the arrays W and WERR respectively.

Syntax

Fortran specification:

```
use armpl_library

subroutine slarrb(N, D, LLD, IFIRST, ILAST, RTOL1, RTOL2, OFFSET, W, WGAP,
                  WERR, WORK, IWORK, PIVMIN, SPDIAM, TWIST, INFO)
```

C specification:

```
#include "armpl.h"

void slarrb_(const armpl_int_t *n, const float *d, const float *lld,
             const armpl_int_t *ifirst, const armpl_int_t *ilast,
             const float *rtol1, const float *rtol2,
             const armpl_int_t *offset, float *w, float *wgap, float *werr,
             float *work, armpl_int_t *iwork, const float *pivmin,
             const float *spdiam, const armpl_int_t *twist,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix.

D Input parameter.

D is REAL

D is an array, dimension (N). The N diagonal elements of the diagonal matrix D.

LLD Input parameter.

LLD is REAL

LLD is an array, dimension (N-1). The (N-1) elements L(i)*L(i)*D(i).

IFIRST Input parameter.

IFIRST is INTEGER

The index of the first eigenvalue to be computed.

ILAST Input parameter.

ILAST is INTEGER

The index of the last eigenvalue to be computed.

RTOL1 Input parameter.

RTOL1 is REAL

RTOL2 Input parameter.

RTOL2 is REAL

Tolerance for the convergence of the bisection intervals. An interval [LEFT,RIGHT] has converged if RIGHT-LEFT.LT.MAX(RTOL1*GAP, RTOL2*MAX(|LEFT|,|RIGHT|)) where GAP is the (estimated) distance to the nearest eigenvalue.

OFFSET Input parameter.

OFFSET is INTEGER

Offset for the arrays W, WGAP and WERR, i.e., the IFIRST-OFFSET through ILAST-OFFSET elements of these arrays are to be used.

W Input and output parameter.

W is REAL

W is an array, dimension (N). On input, W(IFIRST-OFFSET) through W(ILAST-OFFSET) are estimates of the eigenvalues of L D L^T indexed IFIRST through ILAST. On output, these estimates are refined.

WGAP Input and output parameter.

WGAP is REAL

WGAP is an array, dimension (N-1). On input, the (estimated) gaps between consecutive eigenvalues of L D L^T, i.e., WGAP(I-OFFSET) is the gap between eigenvalues I and I+1. Note that if IFIRST.EQ.ILAST then WGAP(IFIRST-OFFSET) must be set to ZERO. On output, these gaps are refined.

WERR Input and output parameter.

WERR is REAL

WERR is an array, dimension (N). On input, WERR(IFIRST-OFFSET) through WERR(ILAST-OFFSET) are the errors in the estimates of the corresponding elements in W. On output, these errors are refined.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (2*N). Workspace.

IWORK Output parameter.

IWORK is INTEGER array, dimension (2*N)

Workspace.

PIVMIN Input parameter.

PIVMIN is REAL

The minimum pivot in the Sturm sequence.

SPDIAM Input parameter.

SPDIAM is REAL

The spectral diameter of the matrix.

TWIST Input parameter.

TWIST is INTEGER

The twist index for the twisted factorization that is used for the negcount. TWIST = N: Compute negcount from L D L^T - LAMBDA I = L+ D+ L+^T TWIST = 1: Compute negcount from L D L^T - LAMBDA I = U- D- U-^T TWIST = R: Compute negcount from L D L^T - LAMBDA I = N(r) D(r) N(r)

INFO Output parameter.

INFO is INTEGER

Error flag.

Related Information

For this routine in other precisions, please see *dlarrb*.

4.17.529 slarrc

Find the number of eigenvalues of the symmetric tridiagonal matrix T that are in the interval (VL,VU] if JOBT = 'T', and of L D L^T if JOBT = 'L'.

Syntax

Fortran specification:

```
use armpl_library

subroutine slarrc(JOBT, N, VL, VU, D, E, PIVMIN, EIGCNT, LCNT, RCNT, INFO)
```

C specification:

```
#include "armpl.h"

void slarrc_(const char *jobt, const armpl_int_t *n, const float *vl,
             const float *vu, const float *d, const float *e,
             const float *pivmin, armpl_int_t *eigcnt, armpl_int_t *lcnt,
             armpl_int_t *rcnt, armpl_int_t *info, ... );
```

Parameters

JOBT Input parameter.

JOBT is CHARACTER*1

= 'T': Compute Sturm count for matrix T. = 'L': Compute Sturm count for matrix L D L^T.

N Input parameter.

N is INTEGER

The order of the matrix. N > 0.

VL Input parameter.

VL is REAL

The lower bound for the eigenvalues.

VU Input parameter.

VU is REAL

The upper bound for the eigenvalues.

D Input parameter.

D is REAL

D is an array, dimension (N). JOBT = 'T': The N diagonal elements of the tridiagonal matrix T. JOBT = 'L': The N diagonal elements of the diagonal matrix D.

E Input parameter.

E is REAL

E is an array, dimension (N). JOBT = 'T': The N-1 offdiagonal elements of the matrix T. JOBT = 'L': The N-1 offdiagonal elements of the matrix L.

PIVMIN Input parameter.

PIVMIN is REAL

The minimum pivot in the Sturm sequence for T.

EIGCNT Output parameter.

EIGCNT is INTEGER

The number of eigenvalues of the symmetric tridiagonal matrix T that are in the interval (VL,VU]

LCNT Output parameter.

LCNT is INTEGER

RCNT Output parameter.

RCNT is INTEGER

The left and right negcounts of the interval.

INFO Output parameter.

INFO is INTEGER

Related Information

For this routine in other precisions, please see *dlarrc*.

4.17.530 slarrd

slarrd computes the eigenvalues of a symmetric tridiagonal matrix T to suitable accuracy. This is an auxiliary code to be called from SSTEMR. The user may ask for all eigenvalues, all eigenvalues in the half-open interval (VL, VU], or the IL-th through IU-th eigenvalues.

To avoid overflow, the matrix must be scaled so that its largest element is no greater than overflow**(1/2) * underflow**(1/4) in absolute value, and for greatest accuracy, it should not be much smaller than that.

See W. Kahan "Accurate Eigenvalues of a Symmetric Tridiagonal Matrix", Report CS41, Computer Science Dept., Stanford University, July 21, 1966.

Syntax

Fortran specification:

```
use armpl_library

subroutine slarrd(RANGE, ORDER, N, VL, VU, IL, IU, GERS, RELTOL, D, E, E2,
                  PIVMIN, NSPLIT, ISPLIT, M, W, WERR, WL, WU, IBLOCK, INDEXW,
                  WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void slarrd_(const char *range, const char *order, const armpl_int_t *n,
             const float *vl, const float *vu, const armpl_int_t *il,
             const armpl_int_t *iu, const float *gers, const float *reltol,
             const float *d, const float *e, const float *e2,
             const float *pivmin, const armpl_int_t *nsplit,
             const armpl_int_t *isplit, armpl_int_t *m, float *w, float *werr,
             float *wl, float *wu, armpl_int_t *iblock, armpl_int_t *indexw,
             float *work, armpl_int_t *iwork, armpl_int_t *info, ... );
```

Parameters

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': ("All") all eigenvalues will be found. = 'V': ("Value") all eigenvalues in the half-open interval (VL, VU] will be found. = 'I': ("Index") the IL-th through IU-th eigenvalues (of the entire matrix) will be found.

ORDER Input parameter.

ORDER is CHARACTER*1

= 'B': ("By Block") the eigenvalues will be grouped by split-off block (see IBLOCK, ISPLIT) and ordered from smallest to largest within the block. = 'E': ("Entire matrix") the eigenvalues for the entire matrix will be ordered from smallest to largest.

N Input parameter.

N is INTEGER

The order of the tridiagonal matrix T. N >= 0.

VL Input parameter.

VL is REAL

If RANGE='V', the lower bound of the interval to be searched for eigenvalues. Eigenvalues less than or equal to VL, or greater than VU, will not be returned. VL < VU. Not referenced if RANGE = 'A' or 'I'.

VU Input parameter.

VU is REAL

If RANGE='V', the upper bound of the interval to be searched for eigenvalues. Eigenvalues less than or equal to VL, or greater than VU, will not be returned. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

GERS Input parameter.

GERS is REAL

GERS is an array, dimension (2*N). The N Gerschgorin intervals (the i-th Gerschgorin interval is (GERS(2*i-1), GERS(2*i)).

RELTOL Input parameter.

RELTOL is REAL

The minimum relative width of an interval. When an interval is narrower than RELTOL times the larger (in magnitude) endpoint, then it is considered to be sufficiently small, i.e., converged. Note: this should always be at least radix*machine epsilon.

D Input parameter.

D is REAL

D is an array, dimension (N). The n diagonal elements of the tridiagonal matrix T.

E Input parameter.

E is REAL

E is an array, dimension (N-1). The (n-1) off-diagonal elements of the tridiagonal matrix T.

E2 Input parameter.

E2 is REAL

E2 is an array, dimension (N-1). The (n-1) squared off-diagonal elements of the tridiagonal matrix T.

PIVMIN Input parameter.

PIVMIN is REAL

The minimum pivot allowed in the Sturm sequence for T.

NSPLIT Input parameter.

NSPLIT is INTEGER

The number of diagonal blocks in the matrix T. 1 <= NSPLIT <= N.

ISPLIT Input parameter.

ISPLIT is INTEGER array, dimension (N)

The splitting points, at which T breaks up into submatrices. The first submatrix consists of rows/columns 1 to ISPLIT(1), the second of rows/columns ISPLIT(1)+1 through ISPLIT(2), etc., and the NSPLIT-th consists of rows/columns ISPLIT(NSPLIT-1)+1 through ISPLIT(NSPLIT)=N. (Only the first NSPLIT elements will actually be used, but since the user cannot know a priori what value NSPLIT will have, N words must be reserved for ISPLIT.)

M Output parameter.

M is INTEGER

The actual number of eigenvalues found. 0 <= M <= N. (See also the description of INFO=2,3.)

W Output parameter.

W is REAL

W is an array, dimension (N). On exit, the first M elements of W will contain the eigenvalue approximations. SLARRD computes an interval I_j = (a_j, b_j] that includes eigenvalue j. The eigenvalue approximation is given as the interval midpoint W(j)= (a_j + b_j)/2. The corresponding error is bounded by WERR(j) = abs(a_j - b_j)/2

WERR Output parameter.

WERR is REAL

WERR is an array, dimension (N). The error bound on the corresponding eigenvalue approximation in W.

WL Output parameter.

WL is REAL

WU Output parameter.

WU is REAL

The interval (WL, WU] contains all the wanted eigenvalues. If RANGE='V', then WL=VL and WU=VU. If RANGE='A', then WL and WU are the global Gerschgorin bounds on the spectrum. If RANGE='I', then WL and WU are computed by SLAEBZ from the index range specified.

IBLOCK Output parameter.

IBLOCK is INTEGER array, dimension (N)

At each row/column j where E(j) is zero or small, the matrix T is considered to split into a block diagonal matrix. On exit, if INFO = 0, IBLOCK(i) specifies to which block (from 1 to the number of blocks) the eigenvalue W(i) belongs. (SLARRD may use the remaining N-M elements as workspace.)

INDEXW Output parameter.

INDEXW is INTEGER array, dimension (N)

The indices of the eigenvalues within each block (submatrix); for example, INDEXW(i)= j and IBLOCK(i)=k imply that the i-th eigenvalue W(i) is the j-th eigenvalue in block k.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (4*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (3*N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: some or all of the eigenvalues failed to converge or were not computed: =1 or 3: Bisection failed to converge for some eigenvalues; these eigenvalues are flagged by a negative block number. The effect is that the eigenvalues may not be as accurate as the absolute and relative tolerances. This is generally caused by unexpectedly inaccurate arithmetic. =2 or 3: RANGE='I' only: Not all of the eigenvalues IL:IU were found. Effect: M < IU+1-IL Cause: non-monotonic arithmetic, causing the Sturm sequence to be non-monotonic. Cure: recalculate, using RANGE='A', and pick out eigenvalues IL:IU. In some cases, increasing the PARAMETER "FUDGE" may make things work. = 4: RANGE='I', and the Gershgorin interval initially used was too small. No eigenvalues were computed. Probable cause: your machine has sloppy floating-point arithmetic. Cure: Increase the PARAMETER "FUDGE", recompile, and try again.

Related Information

For this routine in other precisions, please see *dlarrd*.

4.17.531 slarre

To find the desired eigenvalues of a given real symmetric tridiagonal matrix T, slarre sets any "small" off-diagonal elements to zero, and for each unreduced block T_i, it finds (a) a suitable shift at one end of the block's spectrum, (b) the base representation, T_i - sigma_i I = L_i D_i L_i^T, and (c) eigenvalues of each L_i D_i L_i^T. The representations and eigenvalues found are then used by SSTEMR to compute the eigenvectors of T. The accuracy varies depending on whether bisection is used to find a few eigenvalues or the dqds algorithm (subroutine SLASQ2) to conpute all and then discard any unwanted one. As an added benefit, slarre also outputs the n Gerschgorin intervals for the matrices L_i D_i L_i^T.

Syntax

Fortran specification:

```
use armpl_library

subroutine slarre(RANGE, N, VL, VU, IL, IU, D, E, E2, RTOL1, RTOL2, SPLTOL,
                  NSPLIT, ISPLIT, M, W, WERR, WGAP, IBLOCK, INDEXW, GERS,
                  PIVMIN, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void slarre_(const char *range, const armpl_int_t *n, float *vl, float *vu,
```

(continues on next page)

(continued from previous page)

```
                const armpl_int_t *il, const armpl_int_t *iu, float *d, float *e,
                float *e2, const float *rtol1, const float *rtol2,
                const float *spltol, armpl_int_t *nsplit, armpl_int_t *isplit,
                armpl_int_t *m, float *w, float *werr, float *wgap,
                armpl_int_t *iblock, armpl_int_t *indexw, float *gers,
                float *pivmin, float *work, armpl_int_t *iwork,
                armpl_int_t *info, ... );
```

Parameters

RANGE Input parameter.

RANGE is CHARACTER*1

= 'A': ("All") all eigenvalues will be found. = 'V': ("Value") all eigenvalues in the half-open interval (VL, VU] will be found. = 'I': ("Index") the IL-th through IU-th eigenvalues (of the entire matrix) will be found.

N Input parameter.

N is INTEGER

The order of the matrix. N > 0.

VL Input and output parameter.

VL is REAL

If RANGE='V', the lower bound for the eigenvalues. Eigenvalues less than or equal to VL, or greater than VU, will not be returned. VL < VU. If RANGE='I' or ='A', SLARRE computes bounds on the desired part of the spectrum.

VU Input and output parameter.

VU is REAL

If RANGE='V', the upper bound for the eigenvalues. Eigenvalues less than or equal to VL, or greater than VU, will not be returned. VL < VU. If RANGE='I' or ='A', SLARRE computes bounds on the desired part of the spectrum.

IL Input parameter.

IL is INTEGER

If RANGE='I', the index of the smallest eigenvalue to be returned. 1 <= IL <= IU <= N.

IU Input parameter.

IU is INTEGER

If RANGE='I', the index of the largest eigenvalue to be returned. 1 <= IL <= IU <= N.

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry, the N diagonal elements of the tridiagonal matrix T. On exit, the N diagonal elements of the diagonal matrices D_i.

E Input and output parameter.

E is REAL

E is an array, dimension (N). On entry, the first (N-1) entries contain the subdiagonal elements of the tridiagonal matrix T; E(N) need not be set. On exit, E contains the subdiagonal elements of the unit bidiagonal matrices L_i. The entries E(ISPLIT(I)), 1 <= I <= NSPLIT, contain the base points sigma_i on output.

E2 Input and output parameter.

E2 is REAL

E2 is an array, dimension (N). On entry, the first (N-1) entries contain the SQUARES of the subdiagonal elements of the tridiagonal matrix T; E2(N) need not be set. On exit, the entries E2(ISPLIT(I)), 1 <= I <= NSPLIT, have been set to zero

RTOL1 Input parameter.

RTOL1 is REAL

RTOL2 Input parameter.

RTOL2 is REAL

Parameters for bisection. An interval [LEFT,RIGHT] has converged if RIGHT-LEFT.LT.MAX(RTOL1*GAP, RTOL2*MAX(|LEFT|,|RIGHT|))

SPLTOL Input parameter.

SPLTOL is REAL

The threshold for splitting.

NSPLIT Output parameter.

NSPLIT is INTEGER

The number of blocks T splits into. 1 <= NSPLIT <= N.

ISPLIT Output parameter.

ISPLIT is INTEGER array, dimension (N)

The splitting points, at which T breaks up into blocks. The first block consists of rows/columns 1 to ISPLIT(1), the second of rows/columns ISPLIT(1)+1 through ISPLIT(2), etc., and the NSPLIT-th consists of rows/columns ISPLIT(NSPLIT-1)+1 through ISPLIT(NSPLIT)=N.

M Output parameter.

M is INTEGER

The total number of eigenvalues (of all L_i D_i L_i^T) found.

W Output parameter.

W is REAL

W is an array, dimension (N). The first M elements contain the eigenvalues. The eigenvalues of each of the blocks, L_i D_i L_i^T, are sorted in ascending order (SLARRE may use the remaining N-M elements as workspace).

WERR Output parameter.

WERR is REAL

WERR is an array, dimension (N). The error bound on the corresponding eigenvalue in W.

WGAP Output parameter.

WGAP is REAL

WGAP is an array, dimension (N). The separation from the right neighbor eigenvalue in W. The gap is only with respect to the eigenvalues of the same block as each block has its own representation tree. Exception: at the right end of a block we store the left gap

IBLOCK Output parameter.

IBLOCK is INTEGER array, dimension (N)

The indices of the blocks (submatrices) associated with the corresponding eigenvalues in W; IBLOCK(i)=1 if eigenvalue W(i) belongs to the first block from the top, =2 if W(i) belongs to the second block, etc.

INDEXW Output parameter.

INDEXW is INTEGER array, dimension (N)

The indices of the eigenvalues within each block (submatrix); for example, INDEXW(i)= 10 and IBLOCK(i)=2 imply that the i-th eigenvalue W(i) is the 10-th eigenvalue in block 2

GERS Output parameter.

GERS is REAL

GERS is an array, dimension (2*N). The N Gerschgorin intervals (the i-th Gerschgorin interval is (GERS(2*i-1), GERS(2*i)).

PIVMIN Output parameter.

PIVMIN is REAL

The minimum pivot in the Sturm sequence for T.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (6*N). Workspace.

IWORK Output parameter.

IWORK is INTEGER array, dimension (5*N)

Workspace.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit > 0: A problem occurred in SLARRE. < 0: One of the called subroutines signaled an internal problem. Needs inspection of the corresponding parameter IINFO for further information.

=-1: Problem in SLARRD. = 2: No base representation could be found in MAXTRY iterations. Increasing MAXTRY and recompilation might be a remedy. =-3: Problem in SLARRB when computing the refined root representation for SLASQ2. =-4: Problem in SLARRB when preforming bisection on the desired part of the spectrum. =-5: Problem in SLASQ2. =-6: Problem in SLASQ2.

Related Information

For this routine in other precisions, please see *dlarre*.

4.17.532 slarrf

Given the initial representation L D L^T and its cluster of close eigenvalues (in a relative measure), W(CLSTRT), W(CLSTRT+1), ... W(CLEND), `slarrf` finds a new relatively robust representation L D L^T - SIGMA I = L(+) D(+) L(+)^T such that at least one of the eigenvalues of L(+) D(+) L(+)^T is relatively isolated.

Syntax

Fortran specification:

```
use armpl_library

subroutine slarrf(N, D, L, LD, CLSTRT, CLEND, W, WGAP, WERR, SPDIAM, CLGAPL,
                  CLGAPR, PIVMIN, SIGMA, DPLUS, LPLUS, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void slarrf_(const armpl_int_t *n, const float *d, const float *l,
             const float *ld, const armpl_int_t *clstrt,
             const armpl_int_t *clend, const float *w, float *wgap,
             const float *werr, const float *spdiam, const float *clgapl,
             float *clgapr, const float *pivmin, float *sigma, float *dplus,
             float *lplus, float *work, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix (subblock, if the matrix split).

D Input parameter.

D is REAL

D is an array, dimension (N). The N diagonal elements of the diagonal matrix D.

L Input parameter.

L is REAL

L is an array, dimension (N-1). The (N-1) subdiagonal elements of the unit bidiagonal matrix L.

LD Input parameter.

LD is REAL

LD is an array, dimension (N-1). The (N-1) elements L(i)*D(i).

CLSTRT Input parameter.

CLSTRT is INTEGER

The index of the first eigenvalue in the cluster.

CLEND Input parameter.

CLEND is INTEGER

The index of the last eigenvalue in the cluster.

W Input parameter.

W is REAL

W is an array, dimension. dimension is >= (CLEND-CLSTRT+1) The eigenvalue APPROXIMATIONS of L D L^T in ascending order. W(CLSTRT) through W(CLEND) form the cluster of relatively close eigenalues.

WGAP Input and output parameter.

WGAP is REAL

WGAP is an array, dimension. dimension is >= (CLEND-CLSTRT+1) The separation from the right neighbor eigenvalue in W.

WERR Input parameter.

WERR is REAL

WERR is an array, dimension. dimension is >= (CLEND-CLSTRT+1) WERR contain the semiwidth of the uncertainty interval of the corresponding eigenvalue APPROXIMATION in W

SPDIAM Input parameter.

SPDIAM is REAL

estimate of the spectral diameter obtained from the Gerschgorin intervals

CLGAPL Input parameter.

CLGAPL is REAL

CLGAPR Input parameter.

CLGAPR is REAL

absolute gap on each end of the cluster. Set by the calling routine to protect against shifts too close to eigenvalues outside the cluster.

PIVMIN Input parameter.

PIVMIN is REAL

The minimum pivot allowed in the Sturm sequence.

SIGMA Output parameter.

SIGMA is REAL

The shift used to form L(+) D(+) L(+)^T.

DPLUS Output parameter.

DPLUS is REAL

DPLUS is an array, dimension (N). The N diagonal elements of the diagonal matrix D(+).

LPLUS Output parameter.

LPLUS is REAL

LPLUS is an array, dimension (N-1). The first (N-1) elements of LPLUS contain the subdiagonal elements of the unit bidiagonal matrix L(+).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (2*N). Workspace.

INFO Output parameter.

INFO is INTEGER

Signals processing OK (=0) or failure (=1)

Related Information

For this routine in other precisions, please see *dlarrf*.

4.17.533 slarrj

Given the initial eigenvalue approximations of T, slarrj does bisection to refine the eigenvalues of T, W(IFIRST-OFFSET) through W(ILAST-OFFSET), to more accuracy. Initial guesses for these eigenvalues are input in W, the corresponding estimate of the error in these guesses in WERR. During bisection, intervals [left, right] are maintained by storing their mid-points and semi-widths in the arrays W and WERR respectively.

Syntax

Fortran specification:

```
use armpl_library

subroutine slarrj(N, D, E2, IFIRST, ILAST, RTOL, OFFSET, W, WERR, WORK, IWORK,
                  PIVMIN, SPDIAM, INFO)
```

C specification:

```
#include "armpl.h"

void slarrj_(const armpl_int_t *n, const float *d, const float *e2,
             const armpl_int_t *ifirst, const armpl_int_t *ilast,
             const float *rtol, const armpl_int_t *offset, float *w,
             float *werr, float *work, armpl_int_t *iwork,
             const float *pivmin, const float *spdiam, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix.

D Input parameter.

D is REAL

D is an array, dimension (N). The N diagonal elements of T.

E2 Input parameter.

E2 is REAL

E2 is an array, dimension (N-1). The Squares of the (N-1) subdiagonal elements of T.

IFIRST Input parameter.

IFIRST is INTEGER

The index of the first eigenvalue to be computed.

ILAST Input parameter.

ILAST is INTEGER

The index of the last eigenvalue to be computed.

RTOL Input parameter.

RTOL is REAL

Tolerance for the convergence of the bisection intervals. An interval [LEFT,RIGHT] has converged if RIGHT-LEFT.LT.RTOL*MAX(|LEFT|,|RIGHT|).

OFFSET Input parameter.

OFFSET is INTEGER

Offset for the arrays W and WERR, i.e., the IFIRST-OFFSET through ILAST-OFFSET elements of these arrays are to be used.

W Input and output parameter.

W is REAL

W is an array, dimension (N). On input, W(IFIRST-OFFSET) through W(ILAST-OFFSET) are estimates of the eigenvalues of L D L^T indexed IFIRST through ILAST. On output, these estimates are refined.

WERR Input and output parameter.

WERR is REAL

WERR is an array, dimension (N). On input, WERR(IFIRST-OFFSET) through WERR(ILAST-OFFSET) are the errors in the estimates of the corresponding elements in W. On output, these errors are refined.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (2*N). Workspace.

IWORK Output parameter.

IWORK is INTEGER array, dimension (2*N)

Workspace.

PIVMIN Input parameter.

PIVMIN is REAL

The minimum pivot in the Sturm sequence for T.

SPDIAM Input parameter.

SPDIAM is REAL

The spectral diameter of T.

INFO Output parameter.

INFO is INTEGER

Error flag.

Related Information

For this routine in other precisions, please see *dlarrj*.

4.17.534 slarrk

slarrk computes one eigenvalue of a symmetric tridiagonal matrix T to suitable accuracy. This is an auxiliary code to be called from SSTEMR.

To avoid overflow, the matrix must be scaled so that its largest element is no greater than overflow**(1/2) * underflow**(1/4) in absolute value, and for greatest accuracy, it should not be much smaller than that.

See W. Kahan “Accurate Eigenvalues of a Symmetric Tridiagonal Matrix”, Report CS41, Computer Science Dept., Stanford University, July 21, 1966.

Syntax

Fortran specification:

```
use armpl_library

subroutine slarrk(N, IW, GL, GU, D, E2, PIVMIN, RELTOL, W, WERR, INFO)
```

C specification:

```
#include "armpl.h"

void slarrk_(const armpl_int_t *n, const armpl_int_t *iw, const float *gl,
             const float *gu, const float *d, const float *e2,
             const float *pivmin, const float *reltol, float *w, float *werr,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the tridiagonal matrix T. N >= 0.

IW Input parameter.

IW is INTEGER

The index of the eigenvalues to be returned.

GL Input parameter.

GL is REAL

GU Input parameter.

GU is REAL

An upper and a lower bound on the eigenvalue.

D Input parameter.

D is REAL

D is an array, dimension (N). The n diagonal elements of the tridiagonal matrix T.

E2 Input parameter.

E2 is REAL

E2 is an array, dimension (N-1). The (n-1) squared off-diagonal elements of the tridiagonal matrix T.

PIVMIN Input parameter.

PIVMIN is REAL

The minimum pivot allowed in the Sturm sequence for T.

RELTOL Input parameter.

RELTOL is REAL

The minimum relative width of an interval. When an interval is narrower than RELTOL times the larger (in magnitude) endpoint, then it is considered to be sufficiently small, i.e., converged. Note: this should always be at least radix*machine epsilon.

W Output parameter.

W is REAL

WERR Output parameter.

WERR is REAL

The error bound on the corresponding eigenvalue approximation in W.

INFO Output parameter.

INFO is INTEGER

= 0: Eigenvalue converged = -1: Eigenvalue did NOT converge

Related Information

For this routine in other precisions, please see *dlarrk*.

4.17.535 slarrr

Perform tests to decide whether the symmetric tridiagonal matrix T warrants expensive computations which guarantee high relative accuracy in the eigenvalues.

Syntax

Fortran specification:

```
use armpl_library

subroutine slarrr(N, D, E, INFO)
```

C specification:

```
#include "armpl.h"

void slarrr_(const armpl_int_t *n, const float *d, float *e,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix. N > 0.

D Input parameter.

D is REAL

D is an array, dimension (N). The N diagonal elements of the tridiagonal matrix T.

E Input and output parameter.

E is REAL

E is an array, dimension (N). On entry, the first (N-1) entries contain the subdiagonal elements of the tridiagonal matrix T; E(N) is set to ZERO.

INFO Output parameter.

INFO is INTEGER

INFO = 0(default) : the matrix warrants computations preserving relative accuracy. INFO = 1 : the matrix warrants computations guaranteeing only absolute accuracy.

Related Information

For this routine in other precisions, please see *dlarrr*.

4.17.536 slarrv

slarrv computes the eigenvectors of the tridiagonal matrix T = L D L^T given L, D and APPROXIMATIONS to the eigenvalues of L D L^T . The input eigenvalues should have been computed by SLARRE.

Syntax

Fortran specification:

```
use armpl_library

subroutine slarrv(N, VL, VU, D, L, PIVMIN, ISPLIT, M, DOL, DOU, MINRGP, RTOL1,
                  RTOL2, W, WERR, WGAP, IBLOCK, INDEXW, GERS, Z, LDZ, ISUPPZ,
                  WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void slarrv_(const armpl_int_t *n, const float *vl, const float *vu, float *d,
             float *l, float *pivmin, const armpl_int_t *isplit,
             const armpl_int_t *m, const armpl_int_t *dol,
             const armpl_int_t *dou, const float *minrgp, const float *rtol1,
             const float *rtol2, float *w, float *werr, float *wgap,
             const armpl_int_t *iblock, const armpl_int_t *indexw,
             const float *gers, float *z, const armpl_int_t *ldz,
             armpl_int_t *isuppz, float *work, armpl_int_t *iwork,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix. N >= 0.

VL Input parameter.

VL is REAL

Lower bound of the interval that contains the desired eigenvalues. VL < VU. Needed to compute gaps on the left or right end of the extremal eigenvalues in the desired RANGE.

VU Input parameter.

VU is REAL

Upper bound of the interval that contains the desired eigenvalues. VL < VU. Note: VU is currently not used by this implementation of SLARRV, VU is passed to SLARRV because it could be used compute gaps on the right end of the extremal eigenvalues. However, with not much initial accuracy in LAMBDA and VU, the formula can lead to an overestimation of the right gap and thus to inadequately early RQI 'convergence'. This is currently prevented this by forcing a small right gap. And so it turns out that VU is currently not used by this implementation of SLARRV.

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry, the N diagonal elements of the diagonal matrix D. On exit, D may be overwritten.

L Input and output parameter.

L is REAL

L is an array, dimension (N). On entry, the (N-1) subdiagonal elements of the unit bidiagonal matrix L are in elements 1 to N-1 of L (if the matrix is not split.) At the end of each block is stored the corresponding shift as given by SLARRE. On exit, L is overwritten.

PIVMIN Input parameter.

PIVMIN is REAL

The minimum pivot allowed in the Sturm sequence.

ISPLIT Input parameter.

ISPLIT is INTEGER array, dimension (N)

The splitting points, at which T breaks up into blocks. The first block consists of rows/columns 1 to ISPLIT(1), the second of rows/columns ISPLIT(1)+1 through ISPLIT(2), etc.

M Input parameter.

M is INTEGER

The total number of input eigenvalues. 0 <= M <= N.

DOL Input parameter.

DOL is INTEGER

DOU Input parameter.

DOU is INTEGER

If the user wants to compute only selected eigenvectors from all the eigenvalues supplied, he can specify an index range DOL:DOU. Or else the setting DOL=1, DOU=M should be applied. Note that DOL and DOU refer to the order in which the eigenvalues are stored in W. If the user wants to compute only selected eigenpairs, then the columns DOL-1 to DOU+1 of the eigenvector space Z contain the computed eigenvectors. All other columns of Z are set to zero.

MINRGP Input parameter.

MINRGP is REAL

RTOL1 Input parameter.

RTOL1 is REAL

RTOL2 Input parameter.

RTOL2 is REAL

Parameters for bisection. An interval [LEFT,RIGHT] has converged if RIGHT-LEFT.LT.MAX(RTOL1*GAP, RTOL2*MAX(|LEFT|,|RIGHT|))

W Input and output parameter.

W is REAL

W is an array, dimension (N). The first M elements of W contain the APPROXIMATE eigenvalues for which eigenvectors are to be computed. The eigenvalues should be grouped by split-off block and ordered from smallest to largest within the block (The output array W from SLARRE is expected here). Furthermore, they are with respect to the shift of the corresponding root representation for their block. On exit, W holds the eigenvalues of the UNshifted matrix.

WERR Input and output parameter.

WERR is REAL

WERR is an array, dimension (N). The first M elements contain the semiwidth of the uncertainty interval of the corresponding eigenvalue in W

WGAP Input and output parameter.

WGAP is REAL

WGAP is an array, dimension (N). The separation from the right neighbor eigenvalue in W.

IBLOCK Input parameter.

IBLOCK is INTEGER array, dimension (N)

The indices of the blocks (submatrices) associated with the corresponding eigenvalues in W; IBLOCK(i)=1 if eigenvalue W(i) belongs to the first block from the top, =2 if W(i) belongs to the second block, etc.

INDEXW Input parameter.

INDEXW is INTEGER array, dimension (N)

The indices of the eigenvalues within each block (submatrix); for example, INDEXW(i)= 10 and IBLOCK(i)=2 imply that the i-th eigenvalue W(i) is the 10-th eigenvalue in the second block.

GERS Input parameter.

GERS is REAL

GERS is an array, dimension (2*N). The N Gerschgorin intervals (the i-th Gerschgorin interval is (GERS(2*i-1), GERS(2*i)). The Gerschgorin intervals should be computed from the original UNshifted matrix.

Z Output parameter.

Z is REAL

Z is an array, dimension (LDZ, max(1, M)). If INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix T corresponding to the input eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). Note: the user must ensure that at least max(1, M) columns are supplied in the array Z.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

ISUPPZ Output parameter.

ISUPPZ is INTEGER array, dimension (2*max(1, M))

The support of the eigenvectors in Z, i.e., the indices indicating the nonzero elements in Z. The I-th eigenvector is nonzero only in elements ISUPPZ(2*I-1) through ISUPPZ(2*I).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (12*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (7*N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit

> 0: A problem occurred in SLARRV. < 0: One of the called subroutines signaled an internal problem. Needs inspection of the corresponding parameter IINFO for further information.

=-1: Problem in SLARRB when refining a child's eigenvalues. =-2: Problem in SLARRF when computing the RRR of a child. When a child is inside a tight cluster, it can be difficult to find an RRR. A partial remedy from the user's point of view is to make the parameter MINRGP smaller and recompile. However, as the orthogonality of the computed vectors is proportional to 1/MINRGP, the user should be aware that he might be trading in precision when he decreases MINRGP. =-3: Problem in SLARRB when refining a single eigenvalue after the Rayleigh correction was rejected. = 5: The Rayleigh Quotient Iteration failed to converge to full accuracy in MAXITR steps.

Related Information

For this routine in other precisions, please see *clarrv*, *dlarrv* and *zlarrv*.

4.17.537 slarscl2

SLARSCL2 performs a reciprocal diagonal scaling on an vector:

```
x <-- inv(D) * x
```

where the diagonal matrix D is stored as a vector.

Eventually to be replaced by BLAS_sge_diag_scale in the new BLAS standard.

Syntax

Fortran specification:

```
use armpl_library

subroutine slarscl2(M, N, D, X, LDX)
```

C specification:

```
#include "armpl.h"

void slarscl2_(const armpl_int_t *m, const armpl_int_t *n, const float *d,
               float *x, const armpl_int_t *ldx);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of D and X. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of X. N >= 0.

D Input parameter.

D is REAL

D is an array, length M. Diagonal matrix D, stored as a vector of length M.

X Input and output parameter.

X is REAL

X is an array, dimension (LDX, N). On entry, the vector X to be scaled by D. On exit, the scaled vector.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the vector X. LDX >= M.

Related Information

For this routine in other precisions, please see *clarscl2*, *dlarscl2* and *zlarscl2*.

4.17.538 slartg

`slartg` generate a plane rotation so that

```
[  CS  SN  ]  .  [ F ]  =  [ R ]   where CS**2 + SN**2 = 1.
[ -SN  CS  ]     [ G ]     [ 0 ]
```

This is a slower, more accurate version of the BLAS1 routine SROTG, with the following other differences:

```
F and G are unchanged on return.
If G=0, then CS=1 and SN=0.
If F=0 and (G .ne. 0), then CS=0 and SN=1 without doing any
   floating point operations (saves work in SBDSQR when
   there are zeros on the diagonal).
```

If F exceeds G in magnitude, CS will be positive.

Syntax

Fortran specification:

```
use armpl_library

subroutine slartg(F, G, CS, SN, R)
```

C specification:

```
#include "armpl.h"

void slartg_(const float *f, const float *g, float *cs, float *sn, float *r);
```

Parameters

F Input parameter.

F is REAL

The first component of vector to be rotated.

G Input parameter.

G is REAL

The second component of vector to be rotated.

CS Output parameter.

CS is REAL

The cosine of the rotation.

SN Output parameter.

SN is REAL

The sine of the rotation.

R Output parameter.

R is REAL

The nonzero component of the rotated vector.

This version has a few statements commented out for thread safety (machine parameters are computed on each entry). 10 feb 03, SJH.

Related Information

For this routine in other precisions, please see *clartg*, *dlartg* and *zlartg*.

4.17.539 slartgp

`slartgp` generates a plane rotation so that

```
[  CS  SN  ]  .  [ F ]  =  [ R ]   where CS**2 + SN**2 = 1.
[ -SN  CS  ]     [ G ]     [ 0 ]
```

This is a slower, more accurate version of the Level 1 BLAS routine SROTG, with the following other differences:

```
F and G are unchanged on return.
If G=0, then CS=(+/-)1 and SN=0.
If F=0 and (G .ne. 0), then CS=0 and SN=(+/-)1.
```

The sign is chosen so that R >= 0.

Syntax

Fortran specification:

```
use armpl_library

subroutine slartgp(F, G, CS, SN, R)
```

C specification:

```
#include "armpl.h"

void slartgp_(const float *f, const float *g, float *cs, float *sn,
              float *r);
```

Parameters

F Input parameter.

`F` is `REAL`

The first component of vector to be rotated.

G Input parameter.

`G` is `REAL`

The second component of vector to be rotated.

CS Output parameter.

`CS` is `REAL`

The cosine of the rotation.

SN Output parameter.

`SN` is `REAL`

The sine of the rotation.

R Output parameter.

R is REAL

The nonzero component of the rotated vector.

This version has a few statements commented out for thread safety (machine parameters are computed on each entry). 10 feb 03, SJH.

Related Information

For this routine in other precisions, please see *dlartgp*. It also exists with a native C interface as *LAPACKE_slartgp*.

4.17.540 slartgs

slartgs generates a plane rotation designed to introduce a bulge in Golub-Reinsch-style implicit QR iteration for the bidiagonal SVD problem. X and Y are the top-row entries, and SIGMA is the shift. The computed CS and SN define a plane rotation satisfying

```
[  CS  SN  ]  .  [ X^2 - SIGMA ]  =  [ R ],
[ -SN  CS  ]     [    X * Y    ]     [ 0 ]
```

with R nonnegative. If X^2 - SIGMA and X * Y are 0, then the rotation is by PI/2.

Syntax

Fortran specification:

```
use armpl_library

subroutine slartgs(X, Y, SIGMA, CS, SN)
```

C specification:

```
#include "armpl.h"

void slartgs_(const float *x, const float *y, const float *sigma, float *cs,
              float *sn);
```

Parameters

X Input parameter.

X is REAL

The (1,1) entry of an upper bidiagonal matrix.

Y Input parameter.

Y is REAL

The (1,2) entry of an upper bidiagonal matrix.

SIGMA Input parameter.

SIGMA is REAL

The shift.

CS Output parameter.

CS is REAL

The cosine of the rotation.

SN Output parameter.

SN is REAL

The sine of the rotation.

Related Information

For this routine in other precisions, please see *dlartgs*. It also exists with a native C interface as *LAPACKE_slartgs*.

4.17.541 slartv

slartv applies a vector of real plane rotations to elements of the real vectors x and y. For i = 1,2,...,n

```
( x(i) ) := (  c(i)  s(i) ) ( x(i) )
( y(i) )    ( -s(i)  c(i) ) ( y(i) )
```

Syntax

Fortran specification:

```
use armpl_library

subroutine slartv(N, X, INCX, Y, INCY, C, S, INCC)
```

C specification:

```
#include "armpl.h"

void slartv_(const armpl_int_t *n, float *x, const armpl_int_t *incx,
             float *y, const armpl_int_t *incy, const float *c,
             const float *s, const armpl_int_t *incc);
```

Parameters

N Input parameter.

N is INTEGER

The number of plane rotations to be applied.

X Input and output parameter.

X is REAL

X is an array,. dimension (1+(N-1)*INCX) The vector x.

INCX Input parameter.

INCX is INTEGER

The increment between elements of X. INCX > 0.

Y Input and output parameter.

Y is REAL

Y is an array,. dimension (1+(N-1)*INCY) The vector y.

INCY Input parameter.

INCY is INTEGER

The increment between elements of Y. INCY > 0.

C Input parameter.

C is REAL

C is an array, dimension (1+(N-1)*INCC). The cosines of the plane rotations.

S Input parameter.

S is REAL

S is an array, dimension (1+(N-1)*INCC). The sines of the plane rotations.

INCC Input parameter.

INCC is INTEGER

The increment between elements of C and S. INCC > 0.

Related Information

For this routine in other precisions, please see *clartv*, *dlartv* and *zlartv*.

4.17.542 slaruv

slaruv returns a vector of n random real numbers from a uniform (0,1) distribution (n <= 128).

This is an auxiliary routine called by SLARNV and CLARNV.

Syntax

Fortran specification:

```
use armpl_library

subroutine slaruv(ISEED, N, X)
```

C specification:

```
#include "armpl.h"

void slaruv_(armpl_int_t *iseed, const armpl_int_t *n, float *x);
```

Parameters

ISEED Input and output parameter.

ISEED is INTEGER array, dimension (4)

On entry, the seed of the random number generator; the array elements must be between 0 and 4095, and ISEED(4) must be odd. On exit, the seed is updated.

N Input parameter.

N is INTEGER

The number of random numbers to be generated. N <= 128.

X Output parameter.

X is REAL

X is an array, dimension (N). The generated random numbers.

Related Information

For this routine in other precisions, please see *dlaruv*.

4.17.543 slarz

slarz applies a real elementary reflector H to a real M-by-N matrix C, from either the left or the right. H is represented in the form

```
H = I - tau * v * v**T
```

where tau is a real scalar and v is a real vector.

If tau = 0, then H is taken to be the unit matrix.

H is a product of k elementary reflectors as returned by STZRZF.

Syntax

Fortran specification:

```
use armpl_library

subroutine slarz(SIDE, M, N, L, V, INCV, TAU, C, LDC, WORK)
```

C specification:

```
#include "armpl.h"

void slarz_(const char *side, const armpl_int_t *m, const armpl_int_t *n,
            const armpl_int_t *l, const float *v, const armpl_int_t *incv,
            const float *tau, float *c, const armpl_int_t *ldc, float *work,
            ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': form H * C = 'R': form C * H

M Input parameter.

M is INTEGER

The number of rows of the matrix C.

N Input parameter.

N is INTEGER

The number of columns of the matrix C.

L Input parameter.

L is INTEGER

The number of entries of the vector V containing the meaningful part of the Householder vectors. If SIDE = 'L', M >= L >= 0, if SIDE = 'R', N >= L >= 0.

V Input parameter.

V is REAL

V is an array, dimension (1+(L-1)*abs(INCV)). The vector v in the representation of H as returned by STZRZF. V is not used if TAU = 0.

INCV Input parameter.

INCV is INTEGER

The increment between elements of v. INCV <> 0.

TAU Input parameter.

TAU is REAL

The value tau in the representation of H.

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by the matrix H * C if SIDE = 'L', or C * H if SIDE = 'R'.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension. (N) if SIDE = 'L' or (M) if SIDE = 'R'

Related Information

For this routine in other precisions, please see *clarz*, *dlarz* and *zlarz*.

4.17.544 slarzb

slarzb applies a real block reflector H or its transpose H^T to a real distributed M-by-N C from the left or the right.

Currently, only STOREV = 'R' and DIRECT = 'B' are supported.

Syntax

Fortran specification:

```
use armpl_library

subroutine slarzb(SIDE, TRANS, DIRECT, STOREV, M, N, K, L, V, LDV, T, LDT, C,
                  LDC, WORK, LDWORK)
```

C specification:

```
#include "armpl.h"

void slarzb_(const char *side, const char *trans, const char *direct,
             const char *storev, const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *k, const armpl_int_t *l, const float *v,
             const armpl_int_t *ldv, const float *t, const armpl_int_t *ldt,
             float *c, const armpl_int_t *ldc, float *work,
             const armpl_int_t *ldwork, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply H or H^T from the Left = 'R': apply H or H^T from the Right

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply H (No transpose) = 'C': apply H^T (Transpose)

DIRECT Input parameter.

DIRECT is CHARACTER*1

Indicates how H is formed from a product of elementary reflectors = 'F': H = H(1) H(2) . . . H(k) (Forward, not supported yet) = 'B': H = H(k) . . . H(2) H(1) (Backward)

STOREV Input parameter.

STOREV is CHARACTER*1

Indicates how the vectors which define the elementary reflectors are stored: = 'C': Columnwise (not supported yet) = 'R': Rowwise

M Input parameter.

M is INTEGER

The number of rows of the matrix C.

N Input parameter.

N is INTEGER

The number of columns of the matrix C.

K Input parameter.

K is INTEGER

The order of the matrix T (= the number of elementary reflectors whose product defines the block reflector).

L Input parameter.

L is INTEGER

The number of columns of the matrix V containing the meaningful part of the Householder reflectors. If SIDE = 'L', M >= L >= 0, if SIDE = 'R', N >= L >= 0.

V Input parameter.

V is REAL

V is an array, dimension (LDV,NV).. If STOREV = 'C', NV = K; if STOREV = 'R', NV = L.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. If STOREV = 'C', LDV >= L; if STOREV = 'R', LDV >= K.

T Input parameter.

T is REAL

T is an array, dimension (LDT, K). The triangular K-by-K matrix T in the representation of the block reflector.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= K.

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by H* C or H^T * C or C*H or C*H^T .

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (LDWORK, K) .

LDWORK Input parameter.

LDWORK is INTEGER

The leading dimension of the array WORK. If SIDE = 'L', LDWORK >= max(1, N); if SIDE = 'R', LDWORK >= max(1, M).

Related Information

For this routine in other precisions, please see *clarzb*, *dlarzb* and *zlarzb*.

4.17.545 slarzt

slarzt forms the triangular factor T of a real block reflector H of order > n, which is defined as a product of k elementary reflectors.

If DIRECT = 'F', H = H(1) H(2) . . . H(k) and T is upper triangular;

If DIRECT = 'B', H = H(k) . . . H(2) H(1) and T is lower triangular.

If STOREV = 'C', the vector which defines the elementary reflector H(i) is stored in the i-th column of the array V, and

```
H  =  I - V * T * V**T
```

If STOREV = 'R', the vector which defines the elementary reflector H(i) is stored in the i-th row of the array V, and

```
H  =  I - V**T * T * V
```

Currently, only STOREV = 'R' and DIRECT = 'B' are supported.

Syntax

Fortran specification:

```
use armpl_library

subroutine slarzt(DIRECT, STOREV, N, K, V, LDV, TAU, T, LDT)
```

C specification:

```
#include "armpl.h"

void slarzt_(const char *direct, const char *storev, const armpl_int_t *n,
             const armpl_int_t *k, float *v, const armpl_int_t *ldv,
             const float *tau, float *t, const armpl_int_t *ldt, ... );
```

Parameters

DIRECT Input parameter.

DIRECT is CHARACTER*1

Specifies the order in which the elementary reflectors are multiplied to form the block reflector: = 'F': H = H(1) H(2) . . . H(k) (Forward, not supported yet) = 'B': H = H(k) . . . H(2) H(1) (Backward)

STOREV Input parameter.

STOREV is CHARACTER*1

Specifies how the vectors which define the elementary reflectors are stored (see also Further Details): = 'C': columnwise (not supported yet) = 'R': rowwise

N Input parameter.

N is INTEGER

The order of the block reflector H. N >= 0.

K Input parameter.

K is INTEGER

The order of the triangular factor T (= the number of elementary reflectors). K >= 1.

V Input and output parameter.

V is REAL

V is an array, dimension. (LDV, K) if STOREV = 'C' (LDV, N) if STOREV = 'R' The matrix V. See further details.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. If STOREV = 'C', LDV >= max(1, N); if STOREV = 'R', LDV >= K.

TAU Input parameter.

TAU is REAL

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i).

T Output parameter.

T is REAL

T is an array, dimension (LDT, K). The k by k triangular factor T of the block reflector. If DIRECT = 'F', T is upper triangular; if DIRECT = 'B', T is lower triangular. The rest of the array is not used.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= K.

Related Information

For this routine in other precisions, please see *clarzt*, *dlarzt* and *zlarzt*.

4.17.546 slas2

slas2 computes the singular values of the 2-by-2 matrix

```
[  F   G  ]
[  0   H  ].
```

On return, SSMIN is the smaller singular value and SSMAX is the larger singular value.

Syntax

Fortran specification:

```
use armpl_library

subroutine slas2(F, G, H, SSMIN, SSMAX)
```

C specification:

```
#include "armpl.h"

void slas2_(const float *f, const float *g, const float *h, float *ssmin,
            float *ssmax);
```

Parameters

F Input parameter.

F is REAL

The (1,1) element of the 2-by-2 matrix.

G Input parameter.

G is REAL

The (1,2) element of the 2-by-2 matrix.

H Input parameter.

H is REAL

The (2,2) element of the 2-by-2 matrix.

SSMIN Output parameter.

SSMIN is REAL

The smaller singular value.

SSMAX Output parameter.

SSMAX is REAL

The larger singular value.

Related Information

For this routine in other precisions, please see *dlas2*.

4.17.547 slascl

slascl multiplies the M by N real matrix A by the real scalar CTO/CFROM. This is done without over/underflow as long as the final result CTO*A(I,J)/CFROM does not over/underflow. TYPE specifies that A may be full, upper triangular, lower triangular, upper Hessenberg, or banded.

Syntax

Fortran specification:

```
use armpl_library

subroutine slascl(TYPE, KL, KU, CFROM, CTO, M, N, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void slascl_(const char *type, const armpl_int_t *kl, const armpl_int_t *ku,
             const float *cfrom, const float *cto, const armpl_int_t *m,
             const armpl_int_t *n, float *a, const armpl_int_t *lda,
             armpl_int_t *info, ... );
```

Parameters

TYPE Input parameter.

TYPE is CHARACTER*1

TYPE indices the storage type of the input matrix. = 'G': A is a full matrix. = 'L': A is a lower triangular matrix. = 'U': A is an upper triangular matrix. = 'H': A is an upper Hessenberg matrix. = 'B': A is a symmetric band matrix with lower bandwidth KL and upper bandwidth KU and with the only the lower half stored. = 'Q': A is a symmetric band matrix with lower bandwidth KL and upper bandwidth KU and with the only the upper half stored. = 'Z': A is a band matrix with lower bandwidth KL and upper bandwidth KU. See SGBTRF for storage details.

KL Input parameter.

KL is INTEGER

The lower bandwidth of A. Referenced only if TYPE = 'B', 'Q' or 'Z'.

KU Input parameter.

KU is INTEGER

The upper bandwidth of A. Referenced only if TYPE = 'B', 'Q' or 'Z'.

CFROM Input parameter.

CFROM is REAL

CTO Input parameter.

CTO is REAL

The matrix A is multiplied by CTO/CFROM. A(I,J) is computed without over/underflow if the final result CTO*A(I,J)/CFROM can be represented without over/underflow. CFROM must be nonzero.

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). The matrix to be multiplied by CTO/CFROM. See TYPE for the storage type.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If TYPE = 'G', 'L', 'U', 'H', LDA >= max(1, M); TYPE = 'B', LDA >= KL+1; TYPE = 'Q', LDA >= KU+1; TYPE = 'Z', LDA >= 2*KL+KU+1.

INFO Output parameter.

INFO is INTEGER

0 - successful exit <0 - if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *clascl*, *dlascl* and *zlascl*. It also exists with a native C interface as *LAPACKE_slascl*.

4.17.548 slascl2

slascl2 performs a diagonal scaling on a vector:

```
x <-- D * x
```

where the diagonal matrix D is stored as a vector.

Eventually to be replaced by BLAS_sge_diag_scale in the new BLAS standard.

Syntax

Fortran specification:

```
use armpl_library

subroutine slascl2(M, N, D, X, LDX)
```

C specification:

```
#include "armpl.h"

void slascl2_(const armpl_int_t *m, const armpl_int_t *n, const float *d,
              float *x, const armpl_int_t *ldx);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of D and X. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of X. N >= 0.

D Input parameter.

D is REAL

D is an array, length M. Diagonal matrix D, stored as a vector of length M.

X Input and output parameter.

X is REAL

X is an array, dimension (LDX, N). On entry, the vector X to be scaled by D. On exit, the scaled vector.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the vector X. LDX >= M.

Related Information

For this routine in other precisions, please see *clascl2*, *dlascl2* and *zlascl2*.

4.17.549 slasd0

Using a divide and conquer approach, slasd0 computes the singular value decomposition (SVD) of a real upper bidiagonal N-by-M matrix B with diagonal D and offdiagonal E, where M = N + SQRE. The algorithm computes orthogonal matrices U and VT such that B = U * S * VT. The singular values S are overwritten on D.

A related subroutine, SLASDA, computes only the singular values, and optionally, the singular vectors in compact form.

Syntax

Fortran specification:

```
use armpl_library

subroutine slasd0(N, SQRE, D, E, U, LDU, VT, LDVT, SMLSIZ, IWORK, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void slasd0_(const armpl_int_t *n, const armpl_int_t *sqre, float *d,
             float *e, float *u, const armpl_int_t *ldu, float *vt,
             const armpl_int_t *ldvt, const armpl_int_t *smlsiz,
             armpl_int_t *iwork, float *work, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

On entry, the row dimension of the upper bidiagonal matrix. This is also the dimension of the main diagonal array D.

SQRE Input parameter.

SQRE is INTEGER

Specifies the column dimension of the bidiagonal matrix. = 0: The bidiagonal matrix has column dimension M = N; = 1: The bidiagonal matrix has column dimension M = N+1;

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry D contains the main diagonal of the bidiagonal matrix. On exit D, if INFO = 0, contains its singular values.

E Input and output parameter.

E is REAL

E is an array, dimension (M-1). Contains the subdiagonal entries of the bidiagonal matrix. On exit, E has been destroyed.

U Output parameter.

U is REAL

U is an array, dimension (LDU, N). On exit, U contains the left singular vectors.

LDU Input parameter.

LDU is INTEGER

On entry, leading dimension of U.

VT Output parameter.

VT is REAL

VT is an array, dimension (LDVT, M). On exit, VT^T contains the right singular vectors.

LDVT Input parameter.

LDVT is INTEGER

On entry, leading dimension of VT.

SMLSIZ Input parameter.

SMLSIZ is INTEGER

On entry, maximum size of the subproblems at the bottom of the computation tree.

IWORK Output parameter.

IWORK is INTEGER array, dimension (8*N)

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (3*M2+2*M)** .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = 1, a singular value did not converge

Related Information

For this routine in other precisions, please see *dlasd0*.

4.17.550 slasd1

slasd1 computes the SVD of an upper bidiagonal N-by-M matrix B, where N = NL + NR + 1 and M = N + SQRE. slasd1 is called from SLASD0.

A related subroutine SLASD7 handles the case in which the singular values (and the singular vectors in factored form) are desired.

slasd1 computes the SVD as follows:

```
             ( D1(in)     0     0        0 )
B = U(in) * (   Z1**T     a    Z2**T     b ) * VT(in)
             (   0        0    D2(in)    0 )

  = U(out) * ( D(out) 0) * VT(out)
```

where Z^T = ($Z1^T$ a $Z2^T$ b) = u^T VT^T , and u is a vector of dimension M with ALPHA and BETA in the NL+1 and NL+2 th entries and zeros elsewhere; and the entry b is empty if SQRE = 0.

The left singular vectors of the original matrix are stored in U, and the transpose of the right singular vectors are stored in VT, and the singular values are in D. The algorithm consists of three stages:

```
The first stage consists of deflating the size of the problem
when there are multiple singular values or when there are zeros in
the Z vector.  For each such occurrence the dimension of the
secular equation problem is reduced by one.  This stage is
performed by the routine SLASD2.

The second stage consists of calculating the updated
singular values. This is done by finding the square roots of the
roots of the secular equation via the routine SLASD4 (as called
by SLASD3). This routine also calculates the singular vectors of
the current problem.

The final stage consists of computing the updated singular vectors
directly using the updated singular values.  The singular vectors
for the current problem are multiplied with the singular vectors
from the overall problem.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine slasd1(NL, NR, SQRE, D, ALPHA, BETA, U, LDU, VT, LDVT, IDXQ, IWORK,
                  WORK, INFO)
```

C specification:

```
#include "armpl.h"

void slasd1_(const armpl_int_t *nl, const armpl_int_t *nr,
             const armpl_int_t *sqre, float *d, float *alpha, float *beta,
             float *u, const armpl_int_t *ldu, float *vt,
             const armpl_int_t *ldvt, armpl_int_t *idxq, armpl_int_t *iwork,
             float *work, armpl_int_t *info);
```

Parameters

NL Input parameter.

NL is INTEGER

The row dimension of the upper block. NL >= 1.

NR Input parameter.

NR is INTEGER

The row dimension of the lower block. NR >= 1.

SQRE Input parameter.

SQRE is INTEGER

= 0: the lower block is an NR-by-NR square matrix. = 1: the lower block is an NR-by-(NR+1) rectangular matrix.

The bidiagonal matrix has row dimension N = NL + NR + 1, and column dimension M = N + SQRE.

D Input and output parameter.

D is REAL

D is an array, dimension (NL+NR+1).. N = NL+NR+1 On entry D(1:NL,1:NL) contains the singular values of the upper block; and D(NL+2:N) contains the singular values of the lower block. On exit D(1:N) contains the singular values of the modified matrix.

ALPHA Input and output parameter.

ALPHA is REAL

Contains the diagonal element associated with the added row.

BETA Input and output parameter.

BETA is REAL

Contains the off-diagonal element associated with the added row.

U Input and output parameter.

U is REAL

U is an array, dimension (LDU,N). On entry U(1:NL, 1:NL) contains the left singular vectors of the upper block; U(NL+2:N, NL+2:N) contains the left singular vectors of the lower block. On exit U contains the left singular vectors of the bidiagonal matrix.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U. LDU >= max(1, N).

VT Input and output parameter.

VT is REAL

VT is an array, dimension (LDVT,M). where M = N + SQRE. On entry VT(1:NL+1, 1:NL+1)^T contains the right singular vectors of the upper block; VT(NL+2:M, NL+2:M)^T contains the right singular vectors of the lower block. On exit VT^T contains the right singular vectors of the bidiagonal matrix.

LDVT Input parameter.

LDVT is INTEGER

The leading dimension of the array VT. LDVT >= max(1, M).

IDXQ Input and output parameter.

IDXQ is INTEGER array, dimension (N)

This contains the permutation which will reintegrate the subproblem just solved back into sorted order, i.e. D(IDXQ(I = 1, N)) will be in ascending order.

IWORK Output parameter.

IWORK is INTEGER array, dimension (4*N)

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (3*M2+2*M)** .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = 1, a singular value did not converge

Related Information

For this routine in other precisions, please see *dlasd1*.

4.17.551 slasd2

slasd2 merges the two sets of singular values together into a single sorted set. Then it tries to deflate the size of the problem. There are two ways in which deflation can occur: when two or more singular values are close together or if there is a tiny entry in the Z vector. For each such occurrence the order of the related secular equation problem is reduced by one.

slasd2 is called from SLASD1.

Syntax

Fortran specification:

```
use armpl_library

subroutine slasd2(NL, NR, SQRE, K, D, Z, ALPHA, BETA, U, LDU, VT, LDVT,
                  DSIGMA, U2, LDU2, VT2, LDVT2, IDXP, IDX, IDXC, IDXQ, COLTYP,
                  INFO)
```

C specification:

```
#include "armpl.h"

void slasd2_(const armpl_int_t *nl, const armpl_int_t *nr,
             const armpl_int_t *sqre, armpl_int_t *k, float *d, float *z,
             const float *alpha, const float *beta, float *u,
             const armpl_int_t *ldu, float *vt, const armpl_int_t *ldvt,
             float *dsigma, float *u2, const armpl_int_t *ldu2, float *vt2,
             const armpl_int_t *ldvt2, armpl_int_t *idxp, armpl_int_t *idx,
             armpl_int_t *idxc, armpl_int_t *idxq, armpl_int_t *coltyp,
             armpl_int_t *info);
```

Parameters

NL Input parameter.

NL is INTEGER

The row dimension of the upper block. NL >= 1.

NR Input parameter.

NR is INTEGER

The row dimension of the lower block. NR >= 1.

SQRE Input parameter.

SQRE is INTEGER

= 0: the lower block is an NR-by-NR square matrix. = 1: the lower block is an NR-by-(NR+1) rectangular matrix.

The bidiagonal matrix has N = NL + NR + 1 rows and M = N + SQRE >= N columns.

K Output parameter.

K is INTEGER

Contains the dimension of the non-deflated matrix, This is the order of the related secular equation. 1 <= K <=N.

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry D contains the singular values of the two submatrices to be combined. On exit D contains the trailing (N-K) updated singular values (those which were deflated) sorted into increasing order.

Z Output parameter.

Z is REAL

Z is an array, dimension (N). On exit Z contains the updating row vector in the secular equation.

ALPHA Input parameter.

ALPHA is REAL

Contains the diagonal element associated with the added row.

BETA Input parameter.

BETA is REAL

Contains the off-diagonal element associated with the added row.

U Input and output parameter.

U is REAL

U is an array, dimension (LDU,N). On entry U contains the left singular vectors of two submatrices in the two square blocks with corners at (1,1), (NL, NL), and (NL+2, NL+2), (N,N). On exit U contains the trailing (N-K) updated left singular vectors (those which were deflated) in its last N-K columns.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U. LDU >= N.

VT Input and output parameter.

VT is REAL

VT is an array, dimension (LDVT,M). On entry VT^T contains the right singular vectors of two submatrices in the two square blocks with corners at (1,1), (NL+1, NL+1), and (NL+2, NL+2), (M,M). On exit VT^T contains the trailing (N-K) updated right singular vectors (those which were deflated) in its last N-K columns. In case SQRE =1, the last row of VT spans the right null space.

LDVT Input parameter.

LDVT is INTEGER

The leading dimension of the array VT. LDVT >= M.

DSIGMA Output parameter.

DSIGMA is REAL

DSIGMA is an array, dimension (N). Contains a copy of the diagonal elements (K-1 singular values and one zero) in the secular equation.

U2 Output parameter.

U2 is REAL

U2 is an array, dimension (LDU2,N). Contains a copy of the first K-1 left singular vectors which will be used by SLASD3 in a matrix multiply (SGEMM) to solve for the new left singular vectors. U2 is arranged into four blocks. The first block contains a column with 1 at NL+1 and zero everywhere else; the second block contains non-zero entries only at and above NL; the third contains non-zero entries only below NL+1; and the fourth is dense.

LDU2 Input parameter.

LDU2 is INTEGER

The leading dimension of the array U2. LDU2 >= N.

VT2 Output parameter.

VT2 is REAL

VT2 is an array, dimension (LDVT2,N). $VT2^T$ contains a copy of the first K right singular vectors which will be used by SLASD3 in a matrix multiply (SGEMM) to solve for the new right singular vectors. VT2 is arranged into three blocks. The first block contains a row that corresponds to the special 0 diagonal element in SIGMA; the second block contains non-zeros only at and before NL +1; the third block contains non-zeros only at and after NL +2.

LDVT2 Input parameter.

LDVT2 is INTEGER

The leading dimension of the array VT2. LDVT2 >= M.

IDXP Output parameter.

IDXP is INTEGER array, dimension (N)

This will contain the permutation used to place deflated values of D at the end of the array. On output IDXP(2:K) points to the nondeflated D-values and IDXP(K+1:N) points to the deflated singular values.

IDX Output parameter.

IDX is INTEGER array, dimension (N)

This will contain the permutation used to sort the contents of D into ascending order.

IDXC Output parameter.

IDXC is INTEGER array, dimension (N)

This will contain the permutation used to arrange the columns of the deflated U matrix into three groups: the first group contains non-zero entries only at and above NL, the second contains non-zero entries only below NL+2, and the third is dense.

IDXQ Input and output parameter.

IDXQ is INTEGER array, dimension (N)

This contains the permutation which separately sorts the two sub-problems in D into ascending order. Note that entries in the first hlaf of this permutation must first be moved one position backward; and entries in the second half must first have NL+1 added to their values.

COLTYP Output parameter.

COLTYP is INTEGER array, dimension (N)

As workspace, this will contain a label which will indicate which of the following types a column in the U2 matrix or a row in the VT2 matrix is: 1 : non-zero in the upper half only 2 : non-zero in the lower half only 3 : dense 4 : deflated

On exit, it is an array of dimension 4, with COLTYP(I) being the dimension of the I-th type columns.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *dlasd2*.

4.17.552 slasd3

slasd3 finds all the square roots of the roots of the secular equation, as defined by the values in D and Z. It makes the appropriate calls to SLASD4 and then updates the singular vectors by matrix multiplication.

This code makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray XMP, Cray YMP, Cray C 90, or Cray 2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

slasd3 is called from SLASD1.

Syntax

Fortran specification:

```
use armpl_library

subroutine slasd3(NL, NR, SQRE, K, D, Q, LDQ, DSIGMA, U, LDU, U2, LDU2, VT,
                  LDVT, VT2, LDVT2, IDXC, CTOT, Z, INFO)
```

C specification:

```
#include "armpl.h"

void slasd3_(const armpl_int_t *nl, const armpl_int_t *nr,
             const armpl_int_t *sqre, const armpl_int_t *k, float *d,
             float *q, const armpl_int_t *ldq, float *dsigma, float *u,
             const armpl_int_t *ldu, const float *u2, const armpl_int_t *ldu2,
             float *vt, const armpl_int_t *ldvt, float *vt2,
             const armpl_int_t *ldvt2, const armpl_int_t *idxc,
             const armpl_int_t *ctot, float *z, armpl_int_t *info);
```

Parameters

NL Input parameter.

NL is INTEGER

The row dimension of the upper block. NL >= 1.

NR Input parameter.

NR is INTEGER

The row dimension of the lower block. NR >= 1.

SQRE Input parameter.

SQRE is INTEGER

= 0: the lower block is an NR-by-NR square matrix. = 1: the lower block is an NR-by-(NR+1) rectangular matrix.

The bidiagonal matrix has N = NL + NR + 1 rows and M = N + SQRE >= N columns.

K Input parameter.

K is INTEGER

The size of the secular equation, 1 =< K = < N.

D Output parameter.

D is REAL

D is an array, dimension(K). On exit the square roots of the roots of the secular equation, in ascending order.

Q Output parameter.

Q is REAL

Q is an array, dimension (LDQ, K) .

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= K.

DSIGMA Input and output parameter.

DSIGMA is REAL

DSIGMA is an array, dimension(K). The first K elements of this array contain the old roots of the deflated updating problem. These are the poles of the secular equation.

U Output parameter.

U is REAL

U is an array, dimension (LDU, N). The last N - K columns of this matrix contain the deflated left singular vectors.

LDU Input parameter.

LDU is INTEGER

The leading dimension of the array U. LDU >= N.

U2 Input parameter.

U2 is REAL

U2 is an array, dimension (LDU2, N). The first K columns of this matrix contain the non-deflated left singular vectors for the split problem.

LDU2 Input parameter.

LDU2 is INTEGER

The leading dimension of the array U2. LDU2 >= N.

VT Output parameter.

VT is REAL

VT is an array, dimension (LDVT, M). The last M - K columns of VT^T contain the deflated right singular vectors.

LDVT Input parameter.

LDVT is INTEGER

The leading dimension of the array VT. LDVT >= N.

VT2 Input and output parameter.

VT2 is REAL

VT2 is an array, dimension (LDVT2, N). The first K columns of $VT2^T$ contain the non-deflated right singular vectors for the split problem.

LDVT2 Input parameter.

LDVT2 is INTEGER

The leading dimension of the array VT2. LDVT2 >= N.

IDXC Input parameter.

IDXC is INTEGER array, dimension (N)

The permutation used to arrange the columns of U (and rows of VT) into three groups: the first group contains non-zero entries only at and above (or before) NL +1; the second contains non-zero entries only at and below (or after) NL+2; and the third is dense. The first column of U and the row of VT are treated separately, however.

The rows of the singular vectors found by SLASD4 must be likewise permuted before the matrix multiplies can take place.

CTOT Input parameter.

CTOT is INTEGER array, dimension (4)

A count of the total number of the various types of columns in U (or rows in VT), as described in IDXC. The fourth column type is any column which has been deflated.

Z Input and output parameter.

Z is REAL

Z is an array, dimension (K). The first K elements of this array contain the components of the deflation-adjusted updating row vector.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = 1, a singular value did not converge

Related Information

For this routine in other precisions, please see *dlasd3*.

4.17.553 slasd4

This subroutine computes the square root of the I-th updated eigenvalue of a positive symmetric rank-one modification to a positive diagonal matrix whose entries are given as the squares of the corresponding entries in the array d, and that

```
0 <= D(i) < D(j)  for  i < j
```

and that RHO > 0. This is arranged by the calling routine, and is no loss in generality. The rank-one modified system is thus

```
diag( D ) * diag( D ) +  RHO * Z * Z_transpose.
```

where we assume the Euclidean norm of Z is 1.

The method consists of approximating the rational functions in the secular equation by simpler interpolating rational functions.

Syntax

Fortran specification:

```
use armpl_library

subroutine slasd4(N, I, D, Z, DELTA, RHO, SIGMA, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void slasd4_(const armpl_int_t *n, const armpl_int_t *i, const float *d,
             const float *z, float *delta, const float *rho, float *sigma,
             float *work, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The length of all arrays.

I Input parameter.

I is INTEGER

The index of the eigenvalue to be computed. 1 <= I <= N.

D Input parameter.

D is REAL

D is an array, dimension (N). The original eigenvalues. It is assumed that they are in order, 0 <= D(I) < D(J) for I < J.

Z Input parameter.

Z is REAL

Z is an array, dimension (N). The components of the updating vector.

DELTA Output parameter.

DELTA is REAL

DELTA is an array, dimension (N). If N .ne. 1, DELTA contains (D(j) - sigma_I) in its j-th component. If N = 1, then DELTA(1) = 1. The vector DELTA contains the information necessary to construct the (singular) eigenvectors.

RHO Input parameter.

RHO is REAL

The scalar in the symmetric updating formula.

SIGMA Output parameter.

SIGMA is REAL

The computed sigma_I, the I-th updated eigenvalue.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (N). If N .ne. 1, WORK contains (D(j) + sigma_I) in its j-th component. If N = 1, then WORK(1) = 1.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit > 0: if INFO = 1, the updating process failed.

Related Information

For this routine in other precisions, please see *dlasd4*.

4.17.554 slasd5

This subroutine computes the square root of the I-th eigenvalue of a positive symmetric rank-one modification of a 2-by-2 diagonal matrix

```
diag( D ) * diag( D ) +  RHO * Z * transpose(Z) .
```

The diagonal entries in the array D are assumed to satisfy

```
0 <= D(i) < D(j)  for  i < j .
```

We also assume RHO > 0 and that the Euclidean norm of the vector Z is one.

Syntax

Fortran specification:

```
use armpl_library

subroutine slasd5(I, D, Z, DELTA, RHO, DSIGMA, WORK)
```

C specification:

```
#include "armpl.h"

void slasd5_(const armpl_int_t *i, const float *d, const float *z,
             float *delta, const float *rho, float *dsigma, float *work);
```

Parameters

I Input parameter.

I is INTEGER

The index of the eigenvalue to be computed. I = 1 or I = 2.

D Input parameter.

D is REAL

D is an array, dimension (2). The original eigenvalues. We assume 0 <= D(1) < D(2).

Z Input parameter.

Z is REAL

Z is an array, dimension (2). The components of the updating vector.

DELTA Output parameter.

DELTA is REAL

DELTA is an array, dimension (2). Contains (D(j) - sigma_I) in its j-th component. The vector DELTA contains the information necessary to construct the eigenvectors.

RHO Input parameter.

RHO is REAL

The scalar in the symmetric updating formula.

DSIGMA Output parameter.

DSIGMA is REAL

The computed sigma_I, the I-th updated eigenvalue.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (2). WORK contains (D(j) + sigma_I) in its j-th component.

Related Information

For this routine in other precisions, please see *dlasd5*.

4.17.555 slasd6

slasd6 computes the SVD of an updated upper bidiagonal matrix B obtained by merging two smaller ones by appending a row. This routine is used only for the problem which requires all singular values and optionally singular vector matrices in factored form. B is an N-by-M matrix with N = NL + NR + 1 and M = N + SQRE. A related subroutine, SLASD1, handles the case in which all singular values and singular vectors of the bidiagonal matrix are desired.

slasd6 computes the SVD as follows:

```
              ( D1(in)     0     0        0 )
B = U(in) * (    Z1**T     a    Z2**T     b ) * VT(in)
              (    0        0    D2(in)    0 )

  = U(out) * ( D(out) 0) * VT(out)
```

where Z^T = ($Z1^T$ a $Z2^T$ b) = u^T VT^T , and u is a vector of dimension M with ALPHA and BETA in the NL+1 and NL+2 th entries and zeros elsewhere; and the entry b is empty if SQRE = 0.

The singular values of B can be computed using D1, D2, the first components of all the right singular vectors of the lower block, and the last components of all the right singular vectors of the upper block. These components are stored and updated in VF and VL, respectively, in slasd6. Hence U and VT are not explicitly referenced.

The singular values are stored in D. The algorithm consists of two stages:

```
 The first stage consists of deflating the size of the problem
when there are multiple singular values or if there is a zero
in the Z vector. For each such occurrence the dimension of the
secular equation problem is reduced by one. This stage is
performed by the routine SLASD7.

The second stage consists of calculating the updated
singular values. This is done by finding the roots of the
secular equation via the routine SLASD4 (as called by SLASD8).
This routine also updates VF and VL and computes the distances
between the updated singular values and the old singular
values.
```

slasd6 is called from SLASDA.

Syntax

Fortran specification:

```
use armpl_library

subroutine slasd6(ICOMPQ, NL, NR, SQRE, D, VF, VL, ALPHA, BETA, IDXQ, PERM,
                  GIVPTR, GIVCOL, LDGCOL, GIVNUM, LDGNUM, POLES, DIFL, DIFR,
                  Z, K, C, S, WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void slasd6_(const armpl_int_t *icompq, const armpl_int_t *nl,
             const armpl_int_t *nr, const armpl_int_t *sqre, float *d,
             float *vf, float *vl, float *alpha, float *beta,
             armpl_int_t *idxq, armpl_int_t *perm, armpl_int_t *givptr,
             armpl_int_t *givcol, const armpl_int_t *ldgcol, float *givnum,
             const armpl_int_t *ldgnum, float *poles, float *difl,
             float *difr, float *z, armpl_int_t *k, float *c, float *s,
             float *work, armpl_int_t *iwork, armpl_int_t *info);
```

Parameters

ICOMPQ Input parameter.

ICOMPQ is INTEGER

Specifies whether singular vectors are to be computed in factored form: = 0: Compute singular values only. = 1: Compute singular vectors in factored form as well.

NL Input parameter.

NL is INTEGER

The row dimension of the upper block. NL >= 1.

NR Input parameter.

NR is INTEGER

The row dimension of the lower block. NR >= 1.

SQRE Input parameter.

SQRE is INTEGER

= 0: the lower block is an NR-by-NR square matrix. = 1: the lower block is an NR-by-(NR+1) rectangular matrix.

The bidiagonal matrix has row dimension N = NL + NR + 1, and column dimension M = N + SQRE.

D Input and output parameter.

D is REAL

D is an array, dimension (NL+NR+1).. On entry D(1:NL,1:NL) contains the singular values of the upper block, and D(NL+2:N) contains the singular values of the lower block. On exit D(1:N) contains the singular values of the modified matrix.

VF Input and output parameter.

VF is REAL

VF is an array, dimension (M). On entry, VF(1:NL+1) contains the first components of all right singular vectors of the upper block; and VF(NL+2:M) contains the first components of all right singular vectors of the lower block. On exit, VF contains the first components of all right singular vectors of the bidiagonal matrix.

VL Input and output parameter.

VL is REAL

VL is an array, dimension (M). On entry, VL(1:NL+1) contains the last components of all right singular vectors of the upper block; and VL(NL+2:M) contains the last components of all right singular vectors of the lower block. On exit, VL contains the last components of all right singular vectors of the bidiagonal matrix.

ALPHA Input and output parameter.

ALPHA is REAL

Contains the diagonal element associated with the added row.

BETA Input and output parameter.

BETA is REAL

Contains the off-diagonal element associated with the added row.

IDXQ Input and output parameter.

IDXQ is INTEGER array, dimension (N)

This contains the permutation which will reintegrate the subproblem just solved back into sorted order, i.e. D(IDXQ(I = 1, N)) will be in ascending order.

PERM Output parameter.

PERM is INTEGER array, dimension (N)

The permutations (from deflation and sorting) to be applied to each block. Not referenced if ICOMPQ = 0.

GIVPTR Output parameter.

GIVPTR is INTEGER

The number of Givens rotations which took place in this subproblem. Not referenced if ICOMPQ = 0.

GIVCOL Output parameter.

GIVCOL is INTEGER array, dimension (LDGCOL, 2)

Each pair of numbers indicates a pair of columns to take place in a Givens rotation. Not referenced if ICOMPQ = 0.

LDGCOL Input parameter.

LDGCOL is INTEGER

leading dimension of GIVCOL, must be at least N.

GIVNUM Output parameter.

GIVNUM is REAL

GIVNUM is an array, dimension (LDGNUM, 2). Each number indicates the C or S value to be used in the corresponding Givens rotation. Not referenced if ICOMPQ = 0.

LDGNUM Input parameter.

LDGNUM is INTEGER

The leading dimension of GIVNUM and POLES, must be at least N.

POLES Output parameter.

POLES is REAL

POLES is an array, dimension (LDGNUM, 2). On exit, POLES(1,*) is an array containing the new singular values obtained from solving the secular equation, and POLES(2,*) is an array containing the poles in the secular equation. Not referenced if ICOMPQ = 0.

DIFL Output parameter.

DIFL is REAL

DIFL is an array, dimension (N). On exit, DIFL(I) is the distance between I-th updated (undeflated) singular value and the I-th (undeflated) old singular value.

DIFR Output parameter.

DIFR is REAL

DIFR is an array,. dimension (LDDIFR, 2) if ICOMPQ = 1 and dimension (K) if ICOMPQ = 0. On exit, DIFR(I,1) = D(I) - DSIGMA(I+1), DIFR(K,1) is not defined and will not be referenced.

If ICOMPQ = 1, DIFR(1:K,2) is an array containing the normalizing factors for the right singular vector matrix.

See SLASD8 for details on DIFL and DIFR.

Z Output parameter.

Z is REAL

Z is an array, dimension (M). The first elements of this array contain the components of the deflation-adjusted updating row vector.

K Output parameter.

K is INTEGER

Contains the dimension of the non-deflated matrix, This is the order of the related secular equation. 1 <= K <=N.

C Output parameter.

C is REAL

C contains garbage if SQRE =0 and the C-value of a Givens rotation related to the right null space if SQRE = 1.

S Output parameter.

S is REAL

S contains garbage if SQRE =0 and the S-value of a Givens rotation related to the right null space if SQRE = 1.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (4 * M) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (3 * N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = 1, a singular value did not converge

Related Information

For this routine in other precisions, please see *dlasd6*.

4.17.556 slasd7

slasd7 merges the two sets of singular values together into a single sorted set. Then it tries to deflate the size of the problem. There are two ways in which deflation can occur: when two or more singular values are close together or if there is a tiny entry in the Z vector. For each such occurrence the order of the related secular equation problem is reduced by one.

slasd7 is called from SLASD6.

Syntax

Fortran specification:

```
use armpl_library

subroutine slasd7(ICOMPQ, NL, NR, SQRE, K, D, Z, ZW, VF, VFW, VL, VLW, ALPHA,
                  BETA, DSIGMA, IDX, IDXP, IDXQ, PERM, GIVPTR, GIVCOL, LDGCOL,
                  GIVNUM, LDGNUM, C, S, INFO)
```

C specification:

```
#include "armpl.h"

void slasd7_(const armpl_int_t *icompq, const armpl_int_t *nl,
             const armpl_int_t *nr, const armpl_int_t *sqre, armpl_int_t *k,
             float *d, float *z, float *zw, float *vf, float *vfw, float *vl,
             float *vlw, const float *alpha, const float *beta, float *dsigma,
             armpl_int_t *idx, armpl_int_t *idxp, const armpl_int_t *idxq,
             armpl_int_t *perm, armpl_int_t *givptr, armpl_int_t *givcol,
             const armpl_int_t *ldgcol, float *givnum,
             const armpl_int_t *ldgnum, float *c, float *s,
             armpl_int_t *info);
```

Parameters

ICOMPQ Input parameter.

ICOMPQ is INTEGER

Specifies whether singular vectors are to be computed in compact form, as follows: = 0: Compute singular values only. = 1: Compute singular vectors of upper bidiagonal matrix in compact form.

NL Input parameter.

NL is INTEGER

The row dimension of the upper block. NL >= 1.

NR Input parameter.

NR is INTEGER

The row dimension of the lower block. NR >= 1.

SQRE Input parameter.

SQRE is INTEGER

= 0: the lower block is an NR-by-NR square matrix. = 1: the lower block is an NR-by-(NR+1) rectangular matrix.

The bidiagonal matrix has N = NL + NR + 1 rows and M = N + SQRE >= N columns.

K Output parameter.

K is INTEGER

Contains the dimension of the non-deflated matrix, this is the order of the related secular equation. 1 <= K <=N.

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry D contains the singular values of the two submatrices to be combined. On exit D contains the trailing (N-K) updated singular values (those which were deflated) sorted into increasing order.

Z Output parameter.

Z is REAL

Z is an array, dimension (M). On exit Z contains the updating row vector in the secular equation.

ZW Output parameter.

ZW is REAL

ZW is an array, dimension (M). Workspace for Z.

VF Input and output parameter.

VF is REAL

VF is an array, dimension (M). On entry, VF(1:NL+1) contains the first components of all right singular vectors of the upper block; and VF(NL+2:M) contains the first components of all right singular vectors of the lower block. On exit, VF contains the first components of all right singular vectors of the bidiagonal matrix.

VFW Output parameter.

VFW is REAL

VFW is an array, dimension (M). Workspace for VF.

VL Input and output parameter.

VL is REAL

VL is an array, dimension (M). On entry, VL(1:NL+1) contains the last components of all right singular vectors of the upper block; and VL(NL+2:M) contains the last components of all right singular vectors of the lower block. On exit, VL contains the last components of all right singular vectors of the bidiagonal matrix.

VLW Output parameter.

VLW is REAL

VLW is an array, dimension (M). Workspace for VL.

ALPHA Input parameter.

ALPHA is REAL

Contains the diagonal element associated with the added row.

BETA Input parameter.

BETA is REAL

Contains the off-diagonal element associated with the added row.

DSIGMA Output parameter.

DSIGMA is REAL

DSIGMA is an array, dimension (N). Contains a copy of the diagonal elements (K-1 singular values and one zero) in the secular equation.

IDX Output parameter.

IDX is INTEGER array, dimension (N)

This will contain the permutation used to sort the contents of D into ascending order.

IDXP Output parameter.

IDXP is INTEGER array, dimension (N)

This will contain the permutation used to place deflated values of D at the end of the array. On output IDXP(2:K) points to the nondeflated D-values and IDXP(K+1:N) points to the deflated singular values.

IDXQ Input parameter.

IDXQ is INTEGER array, dimension (N)

This contains the permutation which separately sorts the two sub-problems in D into ascending order. Note that entries in the first half of this permutation must first be moved one position backward; and entries in the second half must first have NL+1 added to their values.

PERM Output parameter.

PERM is INTEGER array, dimension (N)

The permutations (from deflation and sorting) to be applied to each singular block. Not referenced if ICOMPQ = 0.

GIVPTR Output parameter.

GIVPTR is INTEGER

The number of Givens rotations which took place in this subproblem. Not referenced if ICOMPQ = 0.

GIVCOL Output parameter.

GIVCOL is INTEGER array, dimension (LDGCOL, 2)

Each pair of numbers indicates a pair of columns to take place in a Givens rotation. Not referenced if ICOMPQ = 0.

LDGCOL Input parameter.

LDGCOL is INTEGER

The leading dimension of GIVCOL, must be at least N.

GIVNUM Output parameter.

GIVNUM is REAL

GIVNUM is an array, dimension (LDGNUM, 2). Each number indicates the C or S value to be used in the corresponding Givens rotation. Not referenced if ICOMPQ = 0.

LDGNUM Input parameter.

LDGNUM is INTEGER

The leading dimension of GIVNUM, must be at least N.

C Output parameter.

C is REAL

C contains garbage if SQRE =0 and the C-value of a Givens rotation related to the right null space if SQRE = 1.

S Output parameter.

S is REAL

S contains garbage if SQRE =0 and the S-value of a Givens rotation related to the right null space if SQRE = 1.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *dlasd7*.

4.17.557 slasd8

slasd8 finds the square roots of the roots of the secular equation, as defined by the values in DSIGMA and Z. It makes the appropriate calls to SLASD4, and stores, for each element in D, the distance to its two nearest poles (elements in DSIGMA). It also updates the arrays VF and VL, the first and last components of all the right singular vectors of the original bidiagonal matrix.

slasd8 is called from SLASD6.

Syntax

Fortran specification:

```
use armpl_library

subroutine slasd8(ICOMPQ, K, D, Z, VF, VL, DIFL, DIFR, LDDIFR, DSIGMA, WORK,
                  INFO)
```

C specification:

```
#include "armpl.h"

void slasd8_(const armpl_int_t *icompq, const armpl_int_t *k, float *d,
             float *z, float *vf, float *vl, float *difl, float *difr,
             const armpl_int_t *lddifr, float *dsigma, float *work,
             armpl_int_t *info);
```

Parameters

ICOMPQ Input parameter.

ICOMPQ is INTEGER

Specifies whether singular vectors are to be computed in factored form in the calling routine: = 0: Compute singular values only. = 1: Compute singular vectors in factored form as well.

K Input parameter.

K is INTEGER

The number of terms in the rational function to be solved by SLASD4. K >= 1.

D Output parameter.

D is REAL

D is an array, dimension (K). On output, D contains the updated singular values.

Z Input and output parameter.

Z is REAL

Z is an array, dimension (K). On entry, the first K elements of this array contain the components of the deflation-adjusted updating row vector. On exit, Z is updated.

VF Input and output parameter.

VF is REAL

VF is an array, dimension (K). On entry, VF contains information passed through DBEDE8. On exit, VF contains the first K components of the first components of all right singular vectors of the bidiagonal matrix.

VL Input and output parameter.

VL is REAL

VL is an array, dimension (K). On entry, VL contains information passed through DBEDE8. On exit, VL contains the first K components of the last components of all right singular vectors of the bidiagonal matrix.

DIFL Output parameter.

DIFL is REAL

DIFL is an array, dimension (K). On exit, DIFL(I) = D(I) - DSIGMA(I).

DIFR Output parameter.

DIFR is REAL

DIFR is an array,. dimension (LDDIFR, 2) if ICOMPQ = 1 and dimension (K) if ICOMPQ = 0. On exit, DIFR(I,1) = D(I) - DSIGMA(I+1), DIFR(K,1) is not defined and will not be referenced.

If ICOMPQ = 1, DIFR(1:K,2) is an array containing the normalizing factors for the right singular vector matrix.

LDDIFR Input parameter.

LDDIFR is INTEGER

The leading dimension of DIFR, must be at least K.

DSIGMA Input and output parameter.

DSIGMA is REAL

DSIGMA is an array, dimension (K). On entry, the first K elements of this array contain the old roots of the deflated updating problem. These are the poles of the secular equation. On exit, the elements of DSIGMA may be very slightly altered in value.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (3*K) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = 1, a singular value did not converge

Related Information

For this routine in other precisions, please see *dlasd8*.

4.17.558 slasda

Using a divide and conquer approach, slasda computes the singular value decomposition (SVD) of a real upper bidiagonal N-by-M matrix B with diagonal D and offdiagonal E, where M = N + SQRE. The algorithm computes the singular values in the SVD B = U * S * VT. The orthogonal matrices U and VT are optionally computed in compact form.

A related subroutine, SLASD0, computes the singular values and the singular vectors in explicit form.

Syntax

Fortran specification:

```
use armpl_library

subroutine slasda(ICOMPQ, SMLSIZ, N, SQRE, D, E, U, LDU, VT, K, DIFL, DIFR, Z,
                  POLES, GIVPTR, GIVCOL, LDGCOL, PERM, GIVNUM, C, S, WORK,
                  IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void slasda_(const armpl_int_t *icompq, const armpl_int_t *smlsiz,
             const armpl_int_t *n, const armpl_int_t *sqre, float *d,
             const float *e, float *u, const armpl_int_t *ldu, float *vt,
             armpl_int_t *k, float *difl, float *difr, float *z, float *poles,
             armpl_int_t *givptr, armpl_int_t *givcol,
             const armpl_int_t *ldgcol, armpl_int_t *perm, float *givnum,
             float *c, float *s, float *work, armpl_int_t *iwork,
             armpl_int_t *info);
```

Parameters

ICOMPQ Input parameter.

ICOMPQ is INTEGER

Specifies whether singular vectors are to be computed in compact form, as follows = 0: Compute singular values only. = 1: Compute singular vectors of upper bidiagonal matrix in compact form.

SMLSIZ Input parameter.

SMLSIZ is INTEGER

The maximum size of the subproblems at the bottom of the computation tree.

N Input parameter.

N is INTEGER

The row dimension of the upper bidiagonal matrix. This is also the dimension of the main diagonal array D.

SQRE Input parameter.

SQRE is INTEGER

Specifies the column dimension of the bidiagonal matrix. = 0: The bidiagonal matrix has column dimension M = N; = 1: The bidiagonal matrix has column dimension M = N + 1.

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry D contains the main diagonal of the bidiagonal matrix. On exit D, if INFO = 0, contains its singular values.

E Input parameter.

E is REAL

E is an array, dimension (M-1). Contains the subdiagonal entries of the bidiagonal matrix. On exit, E has been destroyed.

U Output parameter.

U is REAL

U is an array,. dimension (LDU, SMLSIZ) if ICOMPQ = 1, and not referenced if ICOMPQ = 0. If ICOMPQ = 1, on exit, U contains the left singular vector matrices of all subproblems at the bottom level.

LDU Input parameter.

LDU is INTEGER, LDU = > N.

The leading dimension of arrays U, VT, DIFL, DIFR, POLES, GIVNUM, and Z.

VT Output parameter.

VT is REAL

VT is an array,. dimension (LDU, SMLSIZ+1) if ICOMPQ = 1, and not referenced if ICOMPQ = 0. If ICOMPQ = 1, on exit, VT^T contains the right singular vector matrices of all subproblems at the bottom level.

K Output parameter.

K is INTEGER array, dimension (N)

if ICOMPQ = 1 and dimension 1 if ICOMPQ = 0. If ICOMPQ = 1, on exit, K(I) is the dimension of the I-th secular equation on the computation tree.

DIFL Output parameter.

DIFL is REAL

DIFL is an array, dimension (LDU, NLVL),. where NLVL = floor(log_2 (N/SMLSIZ))).

DIFR Output parameter.

DIFR is REAL

DIFR is an array,. dimension (LDU, 2 * NLVL) if ICOMPQ = 1 and dimension (N) if ICOMPQ = 0. If ICOMPQ = 1, on exit, DIFL(1:N, I) and DIFR(1:N, 2 * I - 1) record distances between singular values on the I-th level and singular values on the (I -1)-th level, and DIFR(1:N, 2 * I) contains the normalizing factors for the right singular vector matrix. See SLASD8 for details.

Z Output parameter.

Z is REAL

Z is an array,. dimension (LDU, NLVL) if ICOMPQ = 1 and dimension (N) if ICOMPQ = 0. The first K elements of Z(1, I) contain the components of the deflation-adjusted updating row vector for subproblems on the I-th level.

POLES Output parameter.

POLES is REAL

POLES is an array,. dimension (LDU, 2 * NLVL) if ICOMPQ = 1, and not referenced if ICOMPQ = 0. If ICOMPQ = 1, on exit, POLES(1, 2*I - 1) and POLES(1, 2*I) contain the new and old singular values involved in the secular equations on the I-th level.

GIVPTR Output parameter.

GIVPTR is INTEGER array,

dimension (N) if ICOMPQ = 1, and not referenced if ICOMPQ = 0. If ICOMPQ = 1, on exit, GIVPTR(I) records the number of Givens rotations performed on the I-th problem on the computation tree.

GIVCOL Output parameter.

GIVCOL is INTEGER array,

dimension (LDGCOL, 2 * NLVL) if ICOMPQ = 1, and not referenced if ICOMPQ = 0. If ICOMPQ = 1, on exit, for each I, GIVCOL(1, 2 *I - 1) and GIVCOL(1, 2 *I) record the locations of Givens rotations performed on the I-th level on the computation tree.

LDGCOL Input parameter.

LDGCOL is INTEGER, LDGCOL = > N.

The leading dimension of arrays GIVCOL and PERM.

PERM Output parameter.

PERM is INTEGER array, dimension (LDGCOL, NLVL)

if ICOMPQ = 1, and not referenced if ICOMPQ = 0. If ICOMPQ = 1, on exit, PERM(1, I) records permutations done on the I-th level of the computation tree.

GIVNUM Output parameter.

GIVNUM is REAL

GIVNUM is an array,. dimension (LDU, 2 * NLVL) if ICOMPQ = 1, and not referenced if ICOMPQ = 0. If ICOMPQ = 1, on exit, for each I, GIVNUM(1, 2 *I - 1) and GIVNUM(1, 2 *I) record the C- and S- values of Givens rotations performed on the I-th level on the computation tree.

C Output parameter.

C is REAL

C is an array,. dimension (N) if ICOMPQ = 1, and dimension 1 if ICOMPQ = 0. If ICOMPQ = 1 and the I-th subproblem is not square, on exit, C(I) contains the C-value of a Givens rotation related to the right null space of the I-th subproblem.

S Output parameter.

S is REAL

S is an array, dimension (N) if. ICOMPQ = 1, and dimension 1 if ICOMPQ = 0. If ICOMPQ = 1 and the I-th subproblem is not square, on exit, S(I) contains the S-value of a Givens rotation related to the right null space of the I-th subproblem.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension. (6 * N + (SMLSIZ + 1)*(SMLSIZ + 1)).

IWORK Output parameter.

IWORK is INTEGER array, dimension (7*N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = 1, a singular value did not converge

Related Information

For this routine in other precisions, please see *dlasda*.

4.17.559 slasdq

slasdq computes the singular value decomposition (SVD) of a real (upper or lower) bidiagonal matrix with diagonal D and offdiagonal E, accumulating the transformations if desired. Letting B denote the input bidiagonal matrix, the algorithm computes orthogonal matrices Q and P such that B = Q * S * P^T (P^T denotes the transpose of P). The singular values S are overwritten on D.

The input matrix U is changed to U * Q if desired. The input matrix VT is changed to P^T * VT if desired. The input matrix C is changed to Q^T * C if desired.

See "Computing Small Singular Values of Bidiagonal Matrices With Guaranteed High Relative Accuracy," by J. Demmel and W. Kahan, LAPACK Working Note #3, for a detailed description of the algorithm.

Syntax

Fortran specification:

```
use armpl_library

subroutine slasdq(UPLO, SQRE, N, NCVT, NRU, NCC, D, E, VT, LDVT, U, LDU, C,
                  LDC, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void slasdq_(const char *uplo, const armpl_int_t *sqre, const armpl_int_t *n,
             const armpl_int_t *ncvt, const armpl_int_t *nru,
             const armpl_int_t *ncc, float *d, float *e, float *vt,
             const armpl_int_t *ldvt, float *u, const armpl_int_t *ldu,
             float *c, const armpl_int_t *ldc, float *work, armpl_int_t *info,
             ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the input bidiagonal matrix is upper or lower bidiagonal, and whether it is square are not. UPLO = 'U' or 'u' B is upper bidiagonal. UPLO = 'L' or 'l' B is lower bidiagonal.

SQRE Input parameter.

SQRE is INTEGER

= 0: then the input matrix is N-by-N. = 1: then the input matrix is N-by-(N+1) if UPLU = 'U' and (N+1)-by-N if UPLU = 'L'.

The bidiagonal matrix has N = NL + NR + 1 rows and M = N + SQRE >= N columns.

N Input parameter.

N is INTEGER

On entry, N specifies the number of rows and columns in the matrix. N must be at least 0.

NCVT Input parameter.

NCVT is INTEGER

On entry, NCVT specifies the number of columns of the matrix VT. NCVT must be at least 0.

NRU Input parameter.

NRU is INTEGER

On entry, NRU specifies the number of rows of the matrix U. NRU must be at least 0.

NCC Input parameter.

NCC is INTEGER

On entry, NCC specifies the number of columns of the matrix C. NCC must be at least 0.

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry, D contains the diagonal entries of the bidiagonal matrix whose SVD is desired. On normal exit, D contains the singular values in ascending order.

E Input and output parameter.

E is REAL array.

dimension is (N-1) if SQRE = 0 and N if SQRE = 1. On entry, the entries of E contain the offdiagonal entries of the bidiagonal matrix whose SVD is desired. On normal exit, E will contain 0. If the algorithm does not converge, D and E will contain the diagonal and superdiagonal entries of a bidiagonal matrix orthogonally equivalent to the one given as input.

VT Input and output parameter.

VT is REAL

VT is an array, dimension (LDVT, NCVT). On entry, contains a matrix which on exit has been premultiplied by P^T , dimension N-by-NCVT if SQRE = 0 and (N+1)-by-NCVT if SQRE = 1 (not referenced if NCVT=0).

LDVT Input parameter.

LDVT is INTEGER

On entry, LDVT specifies the leading dimension of VT as declared in the calling (sub) program. LDVT must be at least 1. If NCVT is nonzero LDVT must also be at least N.

U Input and output parameter.

U is REAL

U is an array, dimension (LDU, N). On entry, contains a matrix which on exit has been postmultiplied by Q, dimension NRU-by-N if SQRE = 0 and NRU-by-(N+1) if SQRE = 1 (not referenced if NRU=0).

LDU Input parameter.

LDU is INTEGER

On entry, LDU specifies the leading dimension of U as declared in the calling (sub) program. LDU must be at least max(1, NRU) .

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, NCC). On entry, contains an N-by-NCC matrix which on exit has been premultiplied by Q^T dimension N-by-NCC if SQRE = 0 and (N+1)-by-NCC if SQRE = 1 (not referenced if NCC=0).

LDC Input parameter.

LDC is INTEGER

On entry, LDC specifies the leading dimension of C as declared in the calling (sub) program. LDC must be at least 1. If NCC is nonzero, LDC must also be at least N.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (4*N). Workspace. Only referenced if one of NCVT, NRU, or NCC is nonzero, and if N is at least 2.

INFO Output parameter.

INFO is INTEGER

On exit, a value of 0 indicates a successful exit. If INFO < 0, argument number -INFO is illegal. If INFO > 0, the algorithm did not converge, and INFO specifies how many superdiagonals did not converge.

Related Information

For this routine in other precisions, please see *dlasdq*.

4.17.560 slasdt

slasdt creates a tree of subproblems for bidiagonal divide and conquer.

Syntax

Fortran specification:

```
use armpl_library

subroutine slasdt(N, LVL, ND, INODE, NDIML, NDIMR, MSUB)
```

C specification:

```
#include "armpl.h"

void slasdt_(const armpl_int_t *n, armpl_int_t *lvl, armpl_int_t *nd,
             armpl_int_t *inode, armpl_int_t *ndiml, armpl_int_t *ndimr,
             const armpl_int_t *msub);
```

Parameters

N Input parameter.

N is INTEGER

On entry, the number of diagonal elements of the bidiagonal matrix.

LVL Output parameter.

LVL is INTEGER

On exit, the number of levels on the computation tree.

ND Output parameter.

ND is INTEGER

On exit, the number of nodes on the tree.

INODE Output parameter.

INODE is INTEGER array, dimension (N)

On exit, centers of subproblems.

NDIML Output parameter.

NDIML is INTEGER array, dimension (N)

On exit, row dimensions of left children.

NDIMR Output parameter.

NDIMR is INTEGER array, dimension (N)

On exit, row dimensions of right children.

MSUB Input parameter.

MSUB is INTEGER

On entry, the maximum row dimension each subproblem at the bottom of the tree can be of.

Related Information

For this routine in other precisions, please see *dlasdt*.

4.17.561 slaset

slaset initializes an m-by-n matrix A to BETA on the diagonal and ALPHA on the offdiagonals.

Syntax

Fortran specification:

```
use armpl_library

subroutine slaset(UPLO, M, N, ALPHA, BETA, A, LDA)
```

C specification:

```
#include "armpl.h"

void slaset_(const char *uplo, const armpl_int_t *m, const armpl_int_t *n,
             const float *alpha, const float *beta, float *a,
             const armpl_int_t *lda, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies the part of the matrix A to be set. = 'U': Upper triangular part is set; the strictly lower triangular part of A is not changed. = 'L': Lower triangular part is set; the strictly upper triangular part of A is not changed. Otherwise: All of the matrix A is set.

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

ALPHA Input parameter.

ALPHA is REAL

The constant to which the offdiagonal elements are to be set.

BETA Input parameter.

BETA is REAL

The constant to which the diagonal elements are to be set.

A Output parameter.

A is REAL

A is an array, dimension (LDA, N). On exit, the leading m-by-n submatrix of A is set as follows:

if UPLO = 'U', A(i,j) = ALPHA, 1<=i<=j-1, 1<=j<=n, if UPLO = 'L', A(i,j) = ALPHA, j+1<=i<=m, 1<=j<=n, otherwise, A(i,j) = ALPHA, 1<=i<=m, 1<=j<=n, i.ne.j,

and, for all UPLO, A(i,i) = BETA, 1<=i<=min(m,n).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

Related Information

For this routine in other precisions, please see *claset*, *dlaset* and *zlaset*. It also exists with a native C interface as *LAPACKE_slaset*.

4.17.562 slasq1

`slasq1` computes the singular values of a real N-by-N bidiagonal matrix with diagonal D and off-diagonal E. The singular values are computed to high relative accuracy, in the absence of denormalization, underflow and overflow. The algorithm was first presented in

"Accurate singular values and differential qd algorithms" by K. V. Fernando and B. N. Parlett, Numer. Math., Vol-67, No. 2, pp. 191-230, 1994,

and the present implementation is described in "An implementation of the dqds Algorithm (Positive Case)", LAPACK Working Note.

Syntax

Fortran specification:

```
use armpl_library

subroutine slasq1(N, D, E, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void slasq1_(const armpl_int_t *n, float *d, float *e, float *work,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The number of rows and columns in the matrix. N >= 0.

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry, D contains the diagonal elements of the bidiagonal matrix whose SVD is desired. On normal exit, D contains the singular values in decreasing order.

E Input and output parameter.

E is REAL

E is an array, dimension (N). On entry, elements E(1:N-1) contain the off-diagonal elements of the bidiagonal matrix whose SVD is desired. On exit, E is overwritten.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (4*N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: the algorithm failed = 1, a split was marked by a positive value in E = 2, current block of Z not diagonalized after 100*N iterations (in inner while loop) On exit D and E represent a matrix with the same singular values which the calling subroutine could use to finish the computation, or even feed back into SLASQ1 = 3, termination criterion of outer while loop not met (program created more than N unreduced blocks)

Related Information

For this routine in other precisions, please see *dlasq1*.

4.17.563 slasq2

slasq2 computes all the eigenvalues of the symmetric positive definite tridiagonal matrix associated with the qd array Z to high relative accuracy are computed to high relative accuracy, in the absence of denormalization, underflow and overflow.

To see the relation of Z to the tridiagonal matrix, let L be a unit lower bidiagonal matrix with subdiagonals Z(2,4,6,,..) and let U be an upper bidiagonal matrix with 1's above and diagonal Z(1,3,5,,..). The tridiagonal is L*U or, if you prefer, the symmetric tridiagonal to which it is similar.

Note : slasq2 defines a logical variable, IEEE, which is true on machines which follow ieee-754 floating-point standard in their handling of infinities and NaNs, and false otherwise. This variable is passed to SLASQ3.

Syntax

Fortran specification:

```
use armpl_library

subroutine slasq2(N, Z, INFO)
```

C specification:

```
#include "armpl.h"

void slasq2_(const armpl_int_t *n, float *z, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The number of rows and columns in the matrix. N >= 0.

Z Input and output parameter.

Z is REAL

Z is an array, dimension (4* N). On entry Z holds the qd array. On exit, entries 1 to N hold the eigenvalues in decreasing order, Z(2*N+1) holds the trace, and Z(2*N+2) holds the sum of the eigenvalues. If N > 2, then Z(2*N+3) holds the iteration count, Z(2*N+4) holds NDIVS/NIN^2, and Z(2*N+5) holds the percentage of shifts that failed.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if the i-th argument is a scalar and had an illegal value, then INFO = -i, if the i-th argument is an array and the j-entry had an illegal value, then INFO = -(i*100+j) > 0: the algorithm failed = 1, a split was marked by a positive value in E = 2, current block of Z not diagonalized after 100*N iterations (in inner while loop). On exit Z holds a qd array with the same eigenvalues as the given Z. = 3, termination criterion of outer while loop not met (program created more than N unreduced blocks)

Related Information

For this routine in other precisions, please see *dlasq2*.

4.17.564 slasq3

slasq3 checks for deflation, computes a shift (TAU) and calls dqds. In case of failure it changes shifts, and tries again until output is positive.

Syntax

Fortran specification:

```
use armpl_library

subroutine slasq3(I0, N0, Z, PP, DMIN, SIGMA, DESIG, QMAX, NFAIL, ITER, NDIV,
                  IEEE, TTYPE, DMIN1, DMIN2, DN, DN1, DN2, G, TAU)
```

C specification:

```
#include "armpl.h"

void slasq3_(const armpl_int_t *i0, const armpl_int_t *n0, const float *z,
             armpl_int_t *pp, float *dmin, float *sigma, float *desig,
             const float *qmax, armpl_int_t *nfail, armpl_int_t *iter,
             armpl_int_t *ndiv, const armpl_int_t *ieee, armpl_int_t *ttype,
             float *dmin1, float *dmin2, float *dn, float *dn1, float *dn2,
             float *g, float *tau);
```

Parameters

I0 Input parameter.

I0 is INTEGER

First index.

N0 Input and output parameter.

N0 is INTEGER

Last index.

Z Input and output parameter.

Z is REAL

Z is an array, dimension (4* N0). Z holds the qd array.

PP Input and output parameter.

PP is INTEGER

PP=0 for ping, PP=1 for pong. PP=2 indicates that flipping was applied to the Z array and that the initial tests for deflation should not be performed.

DMIN Output parameter.

DMIN is REAL

Minimum value of d.

SIGMA Output parameter.

SIGMA is REAL

Sum of shifts used in current segment.

DESIG Input and output parameter.

DESIG is REAL

Lower order part of SIGMA

QMAX Input parameter.

QMAX is REAL

Maximum value of q.

NFAIL Input and output parameter.

NFAIL is INTEGER

Increment NFAIL by 1 each time the shift was too big.

ITER Input and output parameter.

ITER is INTEGER

Increment ITER by 1 for each iteration.

NDIV Input and output parameter.

NDIV is INTEGER

Increment NDIV by 1 for each division.

IEEE Input parameter.

IEEE is LOGICAL

Flag for IEEE or non IEEE arithmetic (passed to SLASQ5).

TTYPE Input and output parameter.

TTYPE is INTEGER

Shift type.

DMIN1 Input and output parameter.

DMIN1 is REAL

DMIN2 Input and output parameter.

DMIN2 is REAL

DN Input and output parameter.

DN is REAL

DN1 Input and output parameter.

DN1 is REAL

DN2 Input and output parameter.

DN2 is REAL

G Input and output parameter.

G is REAL

TAU Input and output parameter.

TAU is REAL

These are passed as arguments in order to save their values between calls to SLASQ3.

Related Information

For this routine in other precisions, please see *dlasq3*.

4.17.565 slasq4

slasq4 computes an approximation TAU to the smallest eigenvalue using values of d from the previous transform.

Syntax

Fortran specification:

```
use armpl_library

subroutine slasq4(I0, N0, Z, PP, N0IN, DMIN, DMIN1, DMIN2, DN, DN1, DN2, TAU,
                  TTYPE, G)
```

C specification:

```
#include "armpl.h"

void slasq4_(const armpl_int_t *i0, const armpl_int_t *n0, const float *z,
             const armpl_int_t *pp, armpl_int_t *n0in, const float *dmin,
             const float *dmin1, const float *dmin2, const float *dn,
             const float *dn1, const float *dn2, float *tau,
             armpl_int_t *ttype, float *g);
```

Parameters

I0 Input parameter.

I0 is INTEGER

First index.

N0 Input parameter.

N0 is INTEGER

Last index.

Z Input parameter.

Z is REAL

Z is an array, dimension (4* N0). Z holds the qd array.

PP Input parameter.

PP is INTEGER

PP=0 for ping, PP=1 for pong.

N0IN Input parameter.

N0IN is INTEGER

The value of N0 at start of EIGTEST.

DMIN Input parameter.

DMIN is REAL

Minimum value of d.

DMIN1 Input parameter.

DMIN1 is REAL

Minimum value of d, excluding D(N0).

DMIN2 Input parameter.

DMIN2 is REAL

Minimum value of d, excluding D(N0) and D(N0-1).

DN Input parameter.

DN is REAL

d(N)

DN1 Input parameter.

DN1 is REAL

d(N-1)

DN2 Input parameter.

DN2 is REAL

d(N-2)

TAU Output parameter.

TAU is REAL

This is the shift.

TTYPE Output parameter.

TTYPE is INTEGER

Shift type.

G Input and output parameter.

G is REAL

G is passed as an argument in order to save its value between calls to SLASQ4.

Related Information

For this routine in other precisions, please see *dlasq4*.

4.17.566 slasq5

slasq5 computes one dqds transform in ping-pong form, one version for IEEE machines another for non IEEE machines.

Syntax

Fortran specification:

```
use armpl_library

subroutine slasq5(I0, N0, Z, PP, TAU, SIGMA, DMIN, DMIN1, DMIN2, DN, DNM1,
                  DNM2, IEEE, EPS)
```

C specification:

```
#include "armpl.h"

void slasq5_(const armpl_int_t *i0, const armpl_int_t *n0, const float *z,
             const armpl_int_t *pp, const float *tau, const float *sigma,
             float *dmin, float *dmin1, float *dmin2, float *dn, float *dnm1,
             float *dnm2, const armpl_int_t *ieee, const float *eps);
```

Parameters

I0 Input parameter.

I0 is INTEGER

First index.

N0 Input parameter.

N0 is INTEGER

Last index.

Z Input parameter.

Z is REAL

Z is an array, dimension (4*N). Z holds the qd array. EMIN is stored in Z(4*N0) to avoid an extra argument.

PP Input parameter.

PP is INTEGER

PP=0 for ping, PP=1 for pong.

TAU Input parameter.

TAU is REAL

This is the shift.

SIGMA Input parameter.

SIGMA is REAL

This is the accumulated shift up to this step.

DMIN Output parameter.

DMIN is REAL

Minimum value of d.

DMIN1 Output parameter.

DMIN1 is REAL

Minimum value of d, excluding D(N0).

DMIN2 Output parameter.

DMIN2 is REAL

Minimum value of d, excluding D(N0) and D(N0-1).

DN Output parameter.

DN is REAL

d(N0), the last value of d.

DNM1 Output parameter.

DNM1 is REAL

d(N0-1).

DNM2 Output parameter.

DNM2 is REAL

d(N0-2).

IEEE Input parameter.

IEEE is LOGICAL

Flag for IEEE or non IEEE arithmetic.

EPS Input parameter.

EPS is REAL

This is the value of epsilon used.

Related Information

For this routine in other precisions, please see *dlasq5*.

4.17.567 slasq6

slasq6 computes one dqd (shift equal to zero) transform in ping-pong form, with protection against underflow and overflow.

Syntax

Fortran specification:

```
use armpl_library

subroutine slasq6(I0, N0, Z, PP, DMIN, DMIN1, DMIN2, DN, DNM1, DNM2)
```

C specification:

```
#include "armpl.h"

void slasq6_(const armpl_int_t *i0, const armpl_int_t *n0, const float *z,
             const armpl_int_t *pp, float *dmin, float *dmin1, float *dmin2,
             float *dn, float *dnm1, float *dnm2);
```

Parameters

I0 Input parameter.

I0 is INTEGER

First index.

N0 Input parameter.

N0 is INTEGER

Last index.

Z Input parameter.

Z is REAL

Z is an array, dimension (4*N). Z holds the qd array. EMIN is stored in Z(4*N0) to avoid an extra argument.

PP Input parameter.

PP is INTEGER

PP=0 for ping, PP=1 for pong.

DMIN Output parameter.

DMIN is REAL

Minimum value of d.

DMIN1 Output parameter.

DMIN1 is REAL

Minimum value of d, excluding D(N0).

DMIN2 Output parameter.

DMIN2 is REAL

Minimum value of d, excluding D(N0) and D(N0-1).

DN Output parameter.

DN is REAL

d(N0), the last value of d.

DNM1 Output parameter.

DNM1 is REAL

d(N0-1).

DNM2 Output parameter.

DNM2 is REAL

d(N0-2).

Related Information

For this routine in other precisions, please see *dlasq6*.

4.17.568 slasr

`slasr` applies a sequence of plane rotations to a real matrix A, from either the left or the right.

When SIDE = 'L', the transformation takes the form

```
A := P*A
```

and when SIDE = 'R', the transformation takes the form

```
A := A*P**T
```

where P is an orthogonal matrix consisting of a sequence of z plane rotations, with z = M when SIDE = 'L' and z = N when SIDE = 'R', and P^T is the transpose of P.

When DIRECT = 'F' (Forward sequence), then

```
P = P(z-1) * ... * P(2) * P(1)
```

and when DIRECT = 'B' (Backward sequence), then

```
P = P(1) * P(2) * ... * P(z-1)
```

where P(k) is a plane rotation matrix defined by the 2-by-2 rotation

```
R(k) = (  c(k)  s(k) )
     = ( -s(k)  c(k) ).
```

When PIVOT = 'V' (Variable pivot), the rotation is performed for the plane (k,k+1), i.e., P(k) has the form

```
P(k) = (  1                                          )
       (       ...                                   )
       (               1                             )
       (                    c(k)  s(k)               )
       (                   -s(k)  c(k)               )
       (                             1               )
       (                                  ...        )
       (                                         1   )
```

where R(k) appears as a rank-2 modification to the identity matrix in rows and columns k and k+1.

When PIVOT = 'T' (Top pivot), the rotation is performed for the plane (1,k+1), so P(k) has the form

```
P(k) = (  c(k)                    s(k)               )
       (         1                                   )
       (              ...                            )
       (                      1                      )
       ( -s(k)                c(k)                   )
       (                             1               )
       (                                  ...        )
       (                                         1   )
```

where R(k) appears in rows and columns 1 and k+1.

Similarly, when PIVOT = 'B' (Bottom pivot), the rotation is performed for the plane (k,z), giving P(k) the form

```
P(k) = ( 1                                           )
       (       ...                                   )
       (              1                              )
       (                   c(k)                s(k)  )
       (                          1                  )
       (                               ...           )
       (                                    1        )
       (                  -s(k)                c(k)  )
```

where R(k) appears in rows and columns k and z. The rotations are performed without ever forming P(k) explicitly.

Syntax

Fortran specification:

```
use armpl_library

subroutine slasr(SIDE, PIVOT, DIRECT, M, N, C, S, A, LDA)
```

C specification:

```
#include "armpl.h"

void slasr_(const char *side, const char *pivot, const char *direct,
            const armpl_int_t *m, const armpl_int_t *n, const float *c,
            const float *s, float *a, const armpl_int_t *lda, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

Specifies whether the plane rotation matrix P is applied to A on the left or the right. = 'L': Left, compute A := P*A = 'R': Right, compute A:= A*P^T

PIVOT Input parameter.

PIVOT is CHARACTER*1

Specifies the plane for which P(k) is a plane rotation matrix. = 'V': Variable pivot, the plane (k,k+1) = 'T': Top pivot, the plane (1,k+1) = 'B': Bottom pivot, the plane (k,z)

DIRECT Input parameter.

DIRECT is CHARACTER*1

Specifies whether P is a forward or backward sequence of plane rotations. = 'F': Forward, P = P(z-1)*...*P(2)*P(1) = 'B': Backward, P = P(1)*P(2)*...*P(z-1)

M Input parameter.

M is INTEGER

The number of rows of the matrix A. If m <= 1, an immediate return is effected.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. If n <= 1, an immediate return is effected.

C Input parameter.

C is REAL

C is an array, dimension. (M-1) if SIDE = 'L' (N-1) if SIDE = 'R' The cosines c(k) of the plane rotations.

S Input parameter.

S is REAL

S is an array, dimension. (M-1) if SIDE = 'L' (N-1) if SIDE = 'R' The sines s(k) of the plane rotations. The 2-by-2 plane rotation part of the matrix P(k), R(k), has the form R(k) = (c(k) s(k)) (-s(k) c(k)).

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). The M-by-N matrix A. On exit, A is overwritten by P* A if SIDE = 'R' or by A*P^T if SIDE = 'L'.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

Related Information

For this routine in other precisions, please see *clasr*, *dlasr* and *zlasr*.

4.17.569 slasrt

Sort the numbers in D in increasing order (if ID = 'I') or in decreasing order (if ID = 'D').

Use Quick Sort, reverting to Insertion sort on arrays of size <= 20. Dimension of STACK limits N to about 2**32.

Syntax

Fortran specification:

```
use armpl_library

subroutine slasrt(ID, N, D, INFO)
```

C specification:

```
#include "armpl.h"

void slasrt_(const char *id, const armpl_int_t *n, float *d,
             armpl_int_t *info, ... );
```

Parameters

ID Input parameter.

ID is CHARACTER*1

= 'I': sort D in increasing order; = 'D': sort D in decreasing order.

N Input parameter.

N is INTEGER

The length of the array D.

D Input and output parameter.

D is REAL

D is an array, dimension (N). On entry, the array to be sorted. On exit, D has been sorted into increasing order (D(1) <= ... <= D(N)) or into decreasing order (D(1) >= ... >= D(N)), depending on ID.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dlasrt*. It also exists with a native C interface as *LAPACKE_slasrt*.

4.17.570 slassq

slassq returns the values scl and smsq such that

```
( scl**2 )*smsq = x( 1 )**2 +...+ x( n )**2 + ( scale**2 )*sumsq,
```

where x(i) = X(1 + (i - 1)*INCX). The value of sumsq is assumed to be non-negative and scl returns the value

```
scl = max( scale, abs( x( i ) ) ).
```

scale and sumsq must be supplied in SCALE and SUMSQ and scl and smsq are overwritten on SCALE and SUMSQ respectively.

The routine makes only one pass through the vector x.

Syntax

Fortran specification:

```
use armpl_library

subroutine slassq(N, X, INCX, SCALE, SUMSQ)
```

C specification:

```
#include "armpl.h"

void slassq_(const armpl_int_t *n, const float *x, const armpl_int_t *incx,
             float *scale, float *sumsq);
```

Parameters

N Input parameter.

N is INTEGER

The number of elements to be used from the vector X.

X Input parameter.

X is REAL

X is an array, dimension (N). The vector for which a scaled sum of squares is computed. x(i) = X(1 + (i - 1)* INCX), 1 <= i <= n.

INCX Input parameter.

INCX is INTEGER

The increment between successive values of the vector X. INCX > 0.

SCALE Input and output parameter.

SCALE is REAL

On entry, the value scale in the equation above. On exit, SCALE is overwritten with scl , the scaling factor for the sum of squares.

SUMSQ Input and output parameter.

SUMSQ is REAL

On entry, the value sumsq in the equation above. On exit, SUMSQ is overwritten with smsq , the basic sum of squares from which scl has been factored out.

Related Information

For this routine in other precisions, please see *classq*, *dlassq* and *zlassq*. It also exists with a native C interface as *LAPACKE_slassq*.

4.17.571 slasv2

slasv2 computes the singular value decomposition of a 2-by-2 triangular matrix

```
[  F   G  ]
[  0   H  ].
```

On return, abs(SSMAX) is the larger singular value, abs(SSMIN) is the smaller singular value, and (CSL,SNL) and (CSR,SNR) are the left and right singular vectors for abs(SSMAX), giving the decomposition

```
[ CSL  SNL ] [  F   G  ] [ CSR -SNR ]  =  [ SSMAX   0   ]
[-SNL  CSL ] [  0   H  ] [ SNR  CSR ]     [  0    SSMIN ].
```

Syntax

Fortran specification:

```
use armpl_library

subroutine slasv2(F, G, H, SSMIN, SSMAX, SNR, CSR, SNL, CSL)
```

C specification:

```
#include "armpl.h"

void slasv2_(const float *f, const float *g, const float *h, float *ssmin,
             float *ssmax, float *snr, float *csr, float *snl, float *csl);
```

Parameters

F Input parameter.

F is REAL

The (1,1) element of the 2-by-2 matrix.

G Input parameter.

G is REAL

The (1,2) element of the 2-by-2 matrix.

H Input parameter.

H is REAL

The (2,2) element of the 2-by-2 matrix.

SSMIN Output parameter.

SSMIN is REAL

abs(SSMIN) is the smaller singular value.

SSMAX Output parameter.

SSMAX is REAL

abs(SSMAX) is the larger singular value.

SNL Output parameter.

SNL is REAL

CSL Output parameter.

CSL is REAL

The vector (CSL, SNL) is a unit left singular vector for the singular value abs(SSMAX).

SNR Output parameter.

SNR is REAL

CSR Output parameter.

CSR is REAL

The vector (CSR, SNR) is a unit right singular vector for the singular value abs(SSMAX).

Related Information

For this routine in other precisions, please see *dlasv2*.

4.17.572 slaswlq

slaswlq computes a blocked Short-Wide LQ factorization of a M-by-N matrix A, where N >= M: A = L * Q

Syntax

Fortran specification:

```
use armpl_library

subroutine slaswlq(M, N, MB, NB, A, LDA, T, LDT, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void slaswlq_(const armpl_int_t *m, const armpl_int_t *n,
              const armpl_int_t *mb, const armpl_int_t *nb, float *a,
              const armpl_int_t *lda, float *t, const armpl_int_t *ldt,
              float *work, const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= M >= 0.

MB Input parameter.

MB is INTEGER

The row block size to be used in the blocked QR. M >= MB >= 1

NB Input parameter.

NB is INTEGER

The column block size to be used in the blocked QR. NB > M.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and below the diagonal of the array contain the N-by-N lower triangular matrix L; the elements above the diagonal represent Q by the rows of blocked V (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

T Output parameter.

T is REAL

T is an array,. dimension (LDT, N * Number_of_row_blocks) where Number_of_row_blocks = CEIL((N-M)/(NB-M)) The blocked upper triangular block reflectors stored in compact form as a sequence of upper triangular blocks. See Further Details below.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= MB.

WORK Output parameter.

(workspace) REAL

(workspace) is an array, dimension (MAX(1, LWORK)) .

LWORK Input parameter.

The dimension of the array WORK. LWORK >= MB * M.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *claswlq*, *dlaswlq* and *zlaswlq*.

4.17.573 slaswp

slaswp performs a series of row interchanges on the matrix A. One row interchange is initiated for each of rows K1 through K2 of A.

Syntax

Fortran specification:

```
use armpl_library

subroutine slaswp(N, A, LDA, K1, K2, IPIV, INCX)
```

C specification:

```
#include "armpl.h"

void slaswp_(const armpl_int_t *n, float *a, const armpl_int_t *lda,
             const armpl_int_t *k1, const armpl_int_t *k2,
             const armpl_int_t *ipiv, const armpl_int_t *incx);
```

Parameters

N Input parameter.

N is INTEGER

The number of columns of the matrix A.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the matrix of column dimension N to which the row interchanges will be applied. On exit, the permuted matrix.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A.

K1 Input parameter.

K1 is INTEGER

The first element of IPIV for which a row interchange will be done.

K2 Input parameter.

K2 is INTEGER

(K2-K1+1) is the number of elements of IPIV for which a row interchange will be done.

IPIV Input parameter.

IPIV is INTEGER array, dimension (K1+(K2-K1)*abs(INCX))

The vector of pivot indices. Only the elements in positions K1 through K1+(K2-K1)*abs(INCX) of IPIV are accessed. IPIV(K1+(K-K1)*abs(INCX)) = L implies rows K and L are to be interchanged.

INCX Input parameter.

INCX is INTEGER

The increment between successive values of IPIV. If INCX is negative, the pivots are applied in reverse order.

Related Information

For this routine in other precisions, please see *claswp*, *dlaswp* and *zlaswp*. It also exists with a native C interface as *LAPACKE_slaswp*.

4.17.574 slasy2

`slasy2` solves for the N1 by N2 matrix X, 1 <= N1,N2 <= 2, in

```
op(TL)*X + ISGN*X*op(TR) = SCALE*B,
```

where TL is N1 by N1, TR is N2 by N2, B is N1 by N2, and ISGN = 1 or -1. op(T) = T or T^T , where T^T denotes the transpose of T.

Syntax

Fortran specification:

```
use armpl_library

subroutine slasy2(LTRANL, LTRANR, ISGN, N1, N2, TL, LDTL, TR, LDTR, B, LDB,
                  SCALE, X, LDX, XNORM, INFO)
```

C specification:

```
#include "armpl.h"

void slasy2_(const armpl_int_t *ltranl, const armpl_int_t *ltranr,
             const armpl_int_t *isgn, const armpl_int_t *n1,
             const armpl_int_t *n2, const float *tl, const armpl_int_t *ldtl,
             const float *tr, const armpl_int_t *ldtr, const float *b,
             const armpl_int_t *ldb, float *scale, float *x,
             const armpl_int_t *ldx, float *xnorm, armpl_int_t *info);
```

Parameters

LTRANL Input parameter.

LTRANL is LOGICAL

On entry, LTRANL specifies the op(TL): = .FALSE., op(TL) = TL, = .TRUE., op(TL) = TL^T .

LTRANR Input parameter.

LTRANR is LOGICAL

On entry, LTRANR specifies the op(TR): = .FALSE., op(TR) = TR, = .TRUE., op(TR) = TR^T .

ISGN Input parameter.

ISGN is INTEGER

On entry, ISGN specifies the sign of the equation as described before. ISGN may only be 1 or -1.

N1 Input parameter.

N1 is INTEGER

On entry, N1 specifies the order of matrix TL. N1 may only be 0, 1 or 2.

N2 Input parameter.

N2 is INTEGER

On entry, N2 specifies the order of matrix TR. N2 may only be 0, 1 or 2.

TL Input parameter.

TL is REAL

TL is an array, dimension (LDTL,2). On entry, TL contains an N1 by N1 matrix.

LDTL Input parameter.

LDTL is INTEGER

The leading dimension of the matrix TL. LDTL >= max(1, N1).

TR Input parameter.

TR is REAL

TR is an array, dimension (LDTR,2). On entry, TR contains an N2 by N2 matrix.

LDTR Input parameter.

LDTR is INTEGER

The leading dimension of the matrix TR. LDTR >= max(1, N2).

B Input parameter.

B is REAL

B is an array, dimension (LDB,2). On entry, the N1 by N2 matrix B contains the right-hand side of the equation.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the matrix B. LDB >= max(1, N1).

SCALE Output parameter.

SCALE is REAL

On exit, SCALE contains the scale factor. SCALE is chosen less than or equal to 1 to prevent the solution overflowing.

X Output parameter.

X is REAL

X is an array, dimension (LDX,2). On exit, X contains the N1 by N2 solution.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the matrix X. LDX >= max(1, N1).

XNORM Output parameter.

XNORM is REAL

On exit, XNORM is the infinity-norm of the solution.

INFO Output parameter.

INFO is INTEGER

On exit, INFO is set to 0: successful exit. 1: TL and TR have too close eigenvalues, so TL or TR is perturbed to get a nonsingular equation. NOTE: In the interests of speed, this routine does not check the inputs for errors.

Related Information

For this routine in other precisions, please see *dlasy2*.

4.17.575 slasyf

`slasyf` computes a partial factorization of a real symmetric matrix A using the Bunch-Kaufman diagonal pivoting method. The partial factorization has the form:

A = (I U12) (A11 0) (I 0) if UPLO = 'U', or:

```
( 0  U22 ) (  0   D  ) ( U12**T U22**T )
```

A = (L11 0) (D 0) ($L11^T$ $L21^T$) if UPLO = 'L'

```
( L21  I ) (  0  A22 ) (  0       I    )
```

where the order of D is at most NB. The actual order is returned in the argument KB, and is either NB or NB-1, or N if N <= NB.

`slasyf` is an auxiliary routine called by SSYTRF. It uses blocked code (calling Level 3 BLAS) to update the submatrix A11 (if UPLO = 'U') or A22 (if UPLO = 'L').

Syntax

Fortran specification:

```
use armpl_library

subroutine slasyf(UPLO, N, NB, KB, A, LDA, IPIV, W, LDW, INFO)
```

C specification:

```
#include "armpl.h"

void slasyf_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nb,
             armpl_int_t *kb, float *a, const armpl_int_t *lda,
             armpl_int_t *ipiv, float *w, const armpl_int_t *ldw,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NB Input parameter.

NB is INTEGER

The maximum number of columns of the matrix A that should be factored. NB should be at least 2 to allow for 2-by-2 pivot blocks.

KB Output parameter.

KB is INTEGER

The number of columns of A that were actually factored. KB is either NB-1 or NB, or N if N <= NB.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, A contains details of the partial factorization.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D.

If UPLO = 'U': Only the last KB elements of IPIV are set.

If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block.

If UPLO = 'L': Only the first KB elements of IPIV are set.

If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

W Output parameter.

W is REAL

W is an array, dimension (LDW, NB) .

LDW Input parameter.

LDW is INTEGER

The leading dimension of the array W. LDW >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit > 0: if INFO = k, D(k,k) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular.

Related Information

For this routine in other precisions, please see *clasyf*, *dlasyf* and *zlasyf*.

4.17.576 slasyf_aa

DLATRF_AA factorizes a panel of a real symmetric matrix A using the Aasen's algorithm. The panel consists of a set of NB rows of A when UPLO is U, or a set of NB columns when UPLO is L.

In order to factorize the panel, the Aasen's algorithm requires the last row, or column, of the previous panel. The first row, or column, of A is set to be the first row, or column, of an identity matrix, which is used to factorize the first panel.

The resulting J-th row of U, or J-th column of L, is stored in the (J-1)-th row, or column, of A (without the unit diagonals), while the diagonal and subdiagonal of A are overwritten by those of T.

Syntax

Fortran specification:

```
use armpl_library

subroutine slasyf_aa(UPLO, J1, M, NB, A, LDA, IPIV, H, LDH, WORK)
```

C specification:

```
#include "armpl.h"

void slasyf_aa_(const char *uplo, const armpl_int_t *j1, const armpl_int_t *m,
                const armpl_int_t *nb, float *a, const armpl_int_t *lda,
                armpl_int_t *ipiv, float *h, const armpl_int_t *ldh,
                float *work, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

J1 Input parameter.

J1 is INTEGER

The location of the first row, or column, of the panel within the submatrix of A, passed to this routine, e.g., when called by SSYTRF_AA, for the first panel, J1 is 1, while for the remaining panels, J1 is 2.

M Input parameter.

M is INTEGER

The dimension of the submatrix. M >= 0.

NB Input parameter.

NB is INTEGER

The dimension of the panel to be facotorized.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, M) for. the first panel, while dimension (LDA,M+1) for the remaining panels.

On entry, A contains the last row, or column, of the previous panel, and the trailing submatrix of A to be factorized, except for the first panel, only the panel is passed.

On exit, the leading panel is factorized.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

IPIV Output parameter.

IPIV is INTEGER array, dimension (M)

Details of the row and column interchanges, the row and column k were interchanged with the row and column IPIV(k).

H Input and output parameter.

H is REAL workspace, dimension (LDH, NB).

LDH Input parameter.

LDH is INTEGER

The leading dimension of the workspace H. LDH >= max(1, M).

WORK Output parameter.

WORK is REAL workspace, dimension (M).

Related Information

For this routine in other precisions, please see *clasyf_aa*, *dlasyf_aa* and *zlasyf_aa*.

4.17.577 slasyf_rook

SLASYF_ROOK computes a partial factorization of a real symmetric matrix A using the bounded Bunch-Kaufman ("rook") diagonal pivoting method. The partial factorization has the form:

A = (I U12) (A11 0) (I 0) if UPLO = 'U', or:

```
( 0  U22 ) (  0   D  ) ( U12**T U22**T )
```

A = (L11 0) (D 0) ($L11^T$ $L21^T$) if UPLO = 'L'

```
( L21  I ) (  0  A22 ) (  0       I    )
```

where the order of D is at most NB. The actual order is returned in the argument KB, and is either NB or NB-1, or N if N <= NB.

SLASYF_ROOK is an auxiliary routine called by SSYTRF_ROOK. It uses blocked code (calling Level 3 BLAS) to update the submatrix A11 (if UPLO = 'U') or A22 (if UPLO = 'L').

Syntax

Fortran specification:

```
use armpl_library

subroutine slasyf_rook(UPLO, N, NB, KB, A, LDA, IPIV, W, LDW, INFO)
```

C specification:

```
#include "armpl.h"

void slasyf_rook_(const char *uplo, const armpl_int_t *n,
                  const armpl_int_t *nb, armpl_int_t *kb, float *a,
                  const armpl_int_t *lda, armpl_int_t *ipiv, float *w,
                  const armpl_int_t *ldw, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NB Input parameter.

NB is INTEGER

The maximum number of columns of the matrix A that should be factored. NB should be at least 2 to allow for 2-by-2 pivot blocks.

KB Output parameter.

KB is INTEGER

The number of columns of A that were actually factored. KB is either NB-1 or NB, or N if N <= NB.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, A contains details of the partial factorization.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D.

If UPLO = 'U': Only the last KB elements of IPIV are set.

If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k-1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k-1 and -IPIV(k-1) were inerchaged, D(k-1:k,k-1:k) is a 2-by-2 diagonal block.

If UPLO = 'L': Only the first KB elements of IPIV are set.

If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k+1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k+1 and -IPIV(k+1) were inerchaged, D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

W Output parameter.

W is REAL

W is an array, dimension (LDW, NB) .

LDW Input parameter.

LDW is INTEGER

The leading dimension of the array W. LDW >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit > 0: if INFO = k, D(k,k) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular.

Related Information

For this routine in other precisions, please see *clasyf_rook*, *dlasyf_rook* and *zlasyf_rook*.

4.17.578 slatbs

slatbs solves one of the triangular systems

```
A *x = s*b  or  A**T*x = s*b
```

with scaling to prevent overflow, where A is an upper or lower triangular band matrix. Here A^T denotes the transpose of A, x and b are n-element vectors, and s is a scaling factor, usually less than or equal to 1, chosen so that the components of x will be less than the overflow threshold. If the unscaled problem will not cause overflow, the Level 2 BLAS routine STBSV is called. If the matrix A is singular (A(j,j) = 0 for some j), then s is set to 0 and a non-trivial solution to A*x = 0 is returned.

Syntax

Fortran specification:

```
use armpl_library

subroutine slatbs(UPLO, TRANS, DIAG, NORMIN, N, KD, AB, LDAB, X, SCALE, CNORM,
                  INFO)
```

C specification:

```
#include "armpl.h"

void slatbs_(const char *uplo, const char *trans, const char *diag,
             const char *normin, const armpl_int_t *n, const armpl_int_t *kd,
             const float *ab, const armpl_int_t *ldab, float *x, float *scale,
             float *cnorm, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the matrix A is upper or lower triangular. = 'U': Upper triangular = 'L': Lower triangular

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the operation applied to A. = 'N': Solve A * x = s*b (No transpose) = 'T': Solve A^T * x = s*b (Transpose) = 'C': Solve A^T * x = s*b (Conjugate transpose = Transpose)

DIAG Input parameter.

DIAG is CHARACTER*1

Specifies whether or not the matrix A is unit triangular. = 'N': Non-unit triangular = 'U': Unit triangular

NORMIN Input parameter.

NORMIN is CHARACTER*1

Specifies whether CNORM has been set or not. = 'Y': CNORM contains the column norms on entry = 'N': CNORM is not set on entry. On exit, the norms will be computed and stored in CNORM.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of subdiagonals or superdiagonals in the triangular matrix A. KD >= 0.

AB Input parameter.

AB is REAL

AB is an array, dimension (LDAB, N). The upper or lower triangular band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

X Input and output parameter.

X is REAL

X is an array, dimension (N). On entry, the right hand side b of the triangular system. On exit, X is overwritten by the solution vector x.

SCALE Output parameter.

SCALE is REAL

The scaling factor s for the triangular system A * x = s*b or A^T * x = s*b. If SCALE = 0, the matrix A is singular or badly scaled, and the vector x is an exact or approximate solution to A*x = 0.

CNORM Input and output parameter.

CNORM is REAL

CNORM is an array, dimension (N). If NORMIN = 'Y', CNORM is an input argument and CNORM(j) contains the norm of the off-diagonal part of the j-th column of A. If TRANS = 'N', CNORM(j) must be greater than or equal to the infinity-norm, and if TRANS = 'T' or 'C', CNORM(j) must be greater than or equal to the 1-norm.

If NORMIN = 'N', CNORM is an output argument and CNORM(j) returns the 1-norm of the offdiagonal part of the j-th column of A.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value

Related Information

For this routine in other precisions, please see *clatbs*, *dlatbs* and *zlatbs*.

4.17.579 slatdf

slatdf uses the LU factorization of the n-by-n matrix Z computed by SGETC2 and computes a contribution to the reciprocal Dif-estimate by solving Z * x = b for x, and choosing the r.h.s. b such that the norm of x is as large as possible. On entry RHS = b holds the contribution from earlier solved sub-systems, and on return RHS = x.

The factorization of Z returned by SGETC2 has the form Z = P*L*U*Q, where P and Q are permutation matrices. L is lower triangular with unit diagonal elements and U is upper triangular.

Syntax

Fortran specification:

```
use armpl_library

subroutine slatdf(IJOB, N, Z, LDZ, RHS, RDSUM, RDSCAL, IPIV, JPIV)
```

C specification:

```
#include "armpl.h"

void slatdf_(const armpl_int_t *ijob, const armpl_int_t *n, const float *z,
             const armpl_int_t *ldz, float *rhs, float *rdsum, float *rdscal,
             const armpl_int_t *ipiv, const armpl_int_t *jpiv);
```

Parameters

IJOB Input parameter.

IJOB is INTEGER

IJOB = 2: First compute an approximative null-vector e of Z using SGECON, e is normalized and solve for Zx = +-e - f with the sign giving the greater value of 2-norm(x). About 5 times as expensive as Default. IJOB .ne. 2: Local look ahead strategy where all entries of the r.h.s. b is chosen as either +1 or -1 (Default).

N Input parameter.

N is INTEGER

The number of columns of the matrix Z.

Z Input parameter.

Z is REAL

Z is an array, dimension (LDZ, N). On entry, the LU part of the factorization of the n-by-n matrix Z computed by SGETC2: Z = P * L * U * Q

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDA >= max(1, N).

RHS Input and output parameter.

RHS is REAL

RHS is an array, dimension N.. On entry, RHS contains contributions from other subsystems. On exit, RHS contains the solution of the subsystem with entries acoording to the value of IJOB (see above).

RDSUM Input and output parameter.

RDSUM is REAL

On entry, the sum of squares of computed contributions to the Dif-estimate under computation by STGSYL, where the scaling factor RDSCAL (see below) has been factored out. On exit, the corresponding sum of squares updated with the contributions from the current sub-system. If TRANS = 'T' RDSUM is not touched. NOTE: RDSUM only makes sense when STGSY2 is called by STGSYL.

RDSCAL Input and output parameter.

RDSCAL is REAL

On entry, scaling factor used to prevent overflow in RDSUM. On exit, RDSCAL is updated w.r.t. the current contributions in RDSUM. If TRANS = 'T', RDSCAL is not touched. NOTE: RDSCAL only makes sense when STGSY2 is called by STGSYL.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N).

The pivot indices; for 1 <= i <= N, row i of the matrix has been interchanged with row IPIV(i).

JPIV Input parameter.

JPIV is INTEGER array, dimension (N).

The pivot indices; for 1 <= j <= N, column j of the matrix has been interchanged with column JPIV(j).

Related Information

For this routine in other precisions, please see *clatdf*, *dlatdf* and *zlatdf*.

4.17.580 slatps

slatps solves one of the triangular systems

```
A *x = s*b  or  A**T*x = s*b
```

with scaling to prevent overflow, where A is an upper or lower triangular matrix stored in packed form. Here A^T denotes the transpose of A, x and b are n-element vectors, and s is a scaling factor, usually less than or equal to 1, chosen so that the components of x will be less than the overflow threshold. If the unscaled problem will not cause overflow, the Level 2 BLAS routine STPSV is called. If the matrix A is singular (A(j,j) = 0 for some j), then s is set to 0 and a non-trivial solution to A*x = 0 is returned.

Syntax

Fortran specification:

```
use armpl_library

subroutine slatps(UPLO, TRANS, DIAG, NORMIN, N, AP, X, SCALE, CNORM, INFO)
```

C specification:

```
#include "armpl.h"

void slatps_(const char *uplo, const char *trans, const char *diag,
             const char *normin, const armpl_int_t *n, const float *ap,
             float *x, float *scale, float *cnorm, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the matrix A is upper or lower triangular. = 'U': Upper triangular = 'L': Lower triangular

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the operation applied to A. = 'N': Solve A * x = s*b (No transpose) = 'T': Solve A^T * x = s*b (Transpose) = 'C': Solve A^T * x = s*b (Conjugate transpose = Transpose)

DIAG Input parameter.

DIAG is CHARACTER*1

Specifies whether or not the matrix A is unit triangular. = 'N': Non-unit triangular = 'U': Unit triangular

NORMIN Input parameter.

NORMIN is CHARACTER*1

Specifies whether CNORM has been set or not. = 'Y': CNORM contains the column norms on entry = 'N': CNORM is not set on entry. On exit, the norms will be computed and stored in CNORM.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input parameter.

AP is REAL

AP is an array, dimension (N*(N+1)/2). The upper or lower triangular matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

X Input and output parameter.

X is REAL

X is an array, dimension (N). On entry, the right hand side b of the triangular system. On exit, X is overwritten by the solution vector x.

SCALE Output parameter.

SCALE is REAL

The scaling factor s for the triangular system A * x = s*b or A^T * x = s*b. If SCALE = 0, the matrix A is singular or badly scaled, and the vector x is an exact or approximate solution to A*x = 0.

CNORM Input and output parameter.

CNORM is REAL

CNORM is an array, dimension (N). If NORMIN = 'Y', CNORM is an input argument and CNORM(j) contains the norm of the off-diagonal part of the j-th column of A. If TRANS = 'N', CNORM(j) must be greater than or equal to the infinity-norm, and if TRANS = 'T' or 'C', CNORM(j) must be greater than or equal to the 1-norm.

If NORMIN = 'N', CNORM is an output argument and CNORM(j) returns the 1-norm of the offdiagonal part of the j-th column of A.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value

Related Information

For this routine in other precisions, please see *clatps*, *dlatps* and *zlatps*.

4.17.581 slatrd

slatrd reduces NB rows and columns of a real symmetric matrix A to symmetric tridiagonal form by an orthogonal similarity transformation Q^T * A * Q, and returns the matrices V and W which are needed to apply the transformation to the unreduced part of A.

If UPLO = 'U', slatrd reduces the last NB rows and columns of a matrix, of which the upper triangle is supplied; if UPLO = 'L', slatrd reduces the first NB rows and columns of a matrix, of which the lower triangle is supplied.

This is an auxiliary routine called by SSYTRD.

Syntax

Fortran specification:

```
use armpl_library

subroutine slatrd(UPLO, N, NB, A, LDA, E, TAU, W, LDW)
```

C specification:

```
#include "armpl.h"

void slatrd_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nb,
             float *a, const armpl_int_t *lda, float *e, float *tau, float *w,
             const armpl_int_t *ldw, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A.

NB Input parameter.

NB is INTEGER

The number of rows and columns to be reduced.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit: if UPLO = 'U', the last NB columns have been reduced to tridiagonal form, with the diagonal elements overwriting the diagonal elements of A; the elements above the diagonal with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors; if UPLO = 'L', the first NB columns have been reduced to tridiagonal form, with the diagonal elements overwriting the diagonal elements of A; the elements below the diagonal with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= (1, N).

E Output parameter.

E is REAL

E is an array, dimension (N-1). If UPLO = 'U', E(n-nb:n-1) contains the superdiagonal elements of the last NB columns of the reduced matrix; if UPLO = 'L', E(1:nb) contains the subdiagonal elements of the first NB columns of the reduced matrix.

TAU Output parameter.

TAU is REAL

TAU is an array, dimension (N-1). The scalar factors of the elementary reflectors, stored in TAU(n-nb:n-1) if UPLO = 'U', and in TAU(1:nb) if UPLO = 'L'. See Further Details.

W Output parameter.

W is REAL

W is an array, dimension (LDW, NB). The n-by-nb matrix W required to update the unreduced part of A.

LDW Input parameter.

LDW is INTEGER

The leading dimension of the array W. LDW >= max(1, N).

Related Information

For this routine in other precisions, please see *clatrd*, *dlatrd* and *zlatrd*.

4.17.582 slatrs

slatrs solves one of the triangular systems

```
A *x = s*b  or  A**T*x = s*b
```

with scaling to prevent overflow. Here A is an upper or lower triangular matrix, A^T denotes the transpose of A, x and b are n-element vectors, and s is a scaling factor, usually less than or equal to 1, chosen so that the components of x will be less than the overflow threshold. If the unscaled problem will not cause overflow, the Level 2 BLAS routine STRSV is called. If the matrix A is singular (A(j,j) = 0 for some j), then s is set to 0 and a non-trivial solution to A*x = 0 is returned.

Syntax

Fortran specification:

```
use armpl_library

subroutine slatrs(UPLO, TRANS, DIAG, NORMIN, N, A, LDA, X, SCALE, CNORM,
                  INFO)
```

C specification:

```
#include "armpl.h"

void slatrs_(const char *uplo, const char *trans, const char *diag,
             const char *normin, const armpl_int_t *n, const float *a,
             const armpl_int_t *lda, float *x, float *scale, float *cnorm,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the matrix A is upper or lower triangular. = 'U': Upper triangular = 'L': Lower triangular

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the operation applied to A. = 'N': Solve A * x = s*b (No transpose) = 'T': Solve A^T * x = s*b (Transpose) = 'C': Solve A^T * x = s*b (Conjugate transpose = Transpose)

DIAG Input parameter.

DIAG is CHARACTER*1

Specifies whether or not the matrix A is unit triangular. = 'N': Non-unit triangular = 'U': Unit triangular

NORMIN Input parameter.

NORMIN is CHARACTER*1

Specifies whether CNORM has been set or not. = 'Y': CNORM contains the column norms on entry = 'N': CNORM is not set on entry. On exit, the norms will be computed and stored in CNORM.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). The triangular matrix A. If UPLO = 'U', the leading n by n upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n by n lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max (1, N).

X Input and output parameter.

X is REAL

X is an array, dimension (N). On entry, the right hand side b of the triangular system. On exit, X is overwritten by the solution vector x.

SCALE Output parameter.

SCALE is REAL

The scaling factor s for the triangular system A * x = s*b or A^T * x = s*b. If SCALE = 0, the matrix A is singular or badly scaled, and the vector x is an exact or approximate solution to A*x = 0.

CNORM Input and output parameter.

CNORM is REAL

CNORM is an array, dimension (N). If NORMIN = 'Y', CNORM is an input argument and CNORM(j) contains the norm of the off-diagonal part of the j-th column of A. If TRANS = 'N', CNORM(j) must be greater than or equal to the infinity-norm, and if TRANS = 'T' or 'C', CNORM(j) must be greater than or equal to the 1-norm.

If NORMIN = 'N', CNORM is an output argument and CNORM(j) returns the 1-norm of the offdiagonal part of the j-th column of A.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value

Related Information

For this routine in other precisions, please see *clatrs*, *dlatrs* and *zlatrs*.

4.17.583 slatrz

slatrz factors the M-by-(M+L) real upper trapezoidal matrix [A1 A2] = [A(1:M,1:M) A(1:M,N-L+1:N)] as (R 0) * Z, by means of orthogonal transformations. Z is an (M+L)-by-(M+L) orthogonal matrix and, R and A1 are M-by-M upper triangular matrices.

Syntax

Fortran specification:

```
use armpl_library

subroutine slatrz(M, N, L, A, LDA, TAU, WORK)
```

C specification:

```
#include "armpl.h"

void slatrz_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *l,
             float *a, const armpl_int_t *lda, float *tau, float *work);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

L Input parameter.

L is INTEGER

The number of columns of the matrix A containing the meaningful part of the Householder vectors. N-M >= L >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the leading M-by-N upper trapezoidal part of the array A must contain the matrix to be factorized. On exit, the leading M-by-M upper triangular part of A contains the upper triangular matrix R, and elements N-L+1 to N of the first M rows of A, with the array TAU, represent the orthogonal matrix Z as a product of M elementary reflectors.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is REAL

TAU is an array, dimension (M). The scalar factors of the elementary reflectors.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (M) .

Related Information

For this routine in other precisions, please see *clatrz*, *dlatrz* and *zlatrz*.

4.17.584 slatsqr

slatsqr computes a blocked Tall-Skinny QR factorization of an M-by-N matrix A, where M >= N: A = Q * R .

Syntax

Fortran specification:

```
use armpl_library

subroutine slatsqr(M, N, MB, NB, A, LDA, T, LDT, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void slatsqr_(const armpl_int_t *m, const armpl_int_t *n,
              const armpl_int_t *mb, const armpl_int_t *nb, float *a,
              const armpl_int_t *lda, float *t, const armpl_int_t *ldt,
              float *work, const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. M >= N >= 0.

MB Input parameter.

MB is INTEGER

The row block size to be used in the blocked QR. MB > N.

NB Input parameter.

NB is INTEGER

The column block size to be used in the blocked QR. N >= NB >= 1.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the N-by-N upper triangular matrix R; the elements below the diagonal represent Q by the columns of blocked V (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

T Output parameter.

T is REAL

T is an array,. dimension (LDT, N * Number_of_row_blocks) where Number_of_row_blocks = CEIL((M-N)/(MB-N)) The blocked upper triangular block reflectors stored in compact form as a sequence of upper triangular blocks. See Further Details below.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= NB.

WORK Output parameter.

(workspace) REAL

(workspace) is an array, dimension (MAX(1, LWORK)) .

LWORK Input parameter.

The dimension of the array WORK. LWORK >= NB*N.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *clatsqr*, *dlatsqr* and *zlatsqr*.

4.17.585 slauu2

`slauu2` computes the product U * U^T or L^T * L, where the triangular factor U or L is stored in the upper or lower triangular part of the array A.

If UPLO = 'U' or 'u' then the upper triangle of the result is stored, overwriting the factor U in A. If UPLO = 'L' or 'l' then the lower triangle of the result is stored, overwriting the factor L in A.

This is the unblocked form of the algorithm, calling Level 2 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine slauu2(UPLO, N, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void slauu2_(const char *uplo, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the triangular factor stored in the array A is upper or lower triangular: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the triangular factor U or L. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product U * U^T ; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product L^T * L.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value

Related Information

For this routine in other precisions, please see *clauu2*, *dlauu2* and *zlauu2*.

4.17.586 slauum

`slauum` computes the product U * U^T or L^T * L, where the triangular factor U or L is stored in the upper or lower triangular part of the array A.

If UPLO = 'U' or 'u' then the upper triangle of the result is stored, overwriting the factor U in A. If UPLO = 'L' or 'l' then the lower triangle of the result is stored, overwriting the factor L in A.

This is the blocked form of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine slauum(UPLO, N, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void slauum_(const char *uplo, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the triangular factor stored in the array A is upper or lower triangular: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the triangular factor U or L. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product U * U^T ; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product L^T * L.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value

Related Information

For this routine in other precisions, please see *clauum*, *dlauum* and *zlauum*. It also exists with a native C interface as *LAPACKE_slauum*.

4.17.587 sorbdb1

`sorbdb1` simultaneously bidiagonalizes the blocks of a tall and skinny matrix X with orthonomal columns:

```
                         [ B11 ]
[ X11 ]    [ P1 |    ] [  0  ]
[-----] = [---------] [-----] Q1**T .
[ X21 ]    [    | P2 ] [ B21 ]
                         [  0  ]
```

X11 is P-by-Q, and X21 is (M-P)-by-Q. Q must be no larger than P, M-P, or M-Q. Routines SORBDB2, SORBDB3, and SORBDB4 handle cases in which Q is not the minimum dimension.

The orthogonal matrices P1, P2, and Q1 are P-by-P, (M-P)-by-(M-P), and (M-Q)-by-(M-Q), respectively. They are represented implicitly by Householder vectors.

B11 and B12 are Q-by-Q bidiagonal matrices represented implicitly by angles THETA, PHI.

Syntax

Fortran specification:

```
use armpl_library

subroutine sorbdb1(M, P, Q, X11, LDX11, X21, LDX21, THETA, PHI, TAUP1, TAUP2,
                   TAUQ1, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sorbdb1_(const armpl_int_t *m, const armpl_int_t *p,
              const armpl_int_t *q, float *x11, const armpl_int_t *ldx11,
              float *x21, const armpl_int_t *ldx21, float *theta, float *phi,
              float *taup1, float *taup2, float *tauq1, float *work,
              const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows X11 plus the number of rows in X21.

P Input parameter.

P is INTEGER

The number of rows in X11. 0 <= P <= M.

Q Input parameter.

Q is INTEGER

The number of columns in X11 and X21. 0 <= Q <= MIN(P,M-P,M-Q).

X11 Input and output parameter.

X11 is REAL

X11 is an array, dimension (LDX11, Q). On entry, the top block of the matrix X to be reduced. On exit, the columns of tril(X11) specify reflectors for P1 and the rows of triu(X11,1) specify reflectors for Q1.

LDX11 Input parameter.

LDX11 is INTEGER

The leading dimension of X11. LDX11 >= P.

X21 Input and output parameter.

X21 is REAL

X21 is an array, dimension (LDX21, Q). On entry, the bottom block of the matrix X to be reduced. On exit, the columns of tril(X21) specify reflectors for P2.

LDX21 Input parameter.

LDX21 is INTEGER

The leading dimension of X21. LDX21 >= M-P.

THETA Output parameter.

THETA is REAL

THETA is an array, dimension (Q). The entries of the bidiagonal blocks B11, B21 are defined by THETA and PHI. See Further Details.

PHI Output parameter.

PHI is REAL

PHI is an array, dimension (Q-1). The entries of the bidiagonal blocks B11, B21 are defined by THETA and PHI. See Further Details.

TAUP1 Output parameter.

TAUP1 is REAL

TAUP1 is an array, dimension (P). The scalar factors of the elementary reflectors that define P1.

TAUP2 Output parameter.

TAUP2 is REAL

TAUP2 is an array, dimension (M-P). The scalar factors of the elementary reflectors that define P2.

TAUQ1 Output parameter.

TAUQ1 is REAL

TAUQ1 is an array, dimension (Q). The scalar factors of the elementary reflectors that define Q1.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= M-Q.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *dorbdb1*.

4.17.588 sorbdb2

sorbdb2 simultaneously bidiagonalizes the blocks of a tall and skinny matrix X with orthonomal columns:

```
                         [ B11 ]
[ X11 ]   [ P1 |    ] [  0  ]
[-----] = [---------] [-----] Q1**T .
[ X21 ]   [    | P2 ] [ B21 ]
                         [  0  ]
```

X11 is P-by-Q, and X21 is (M-P)-by-Q. P must be no larger than M-P, Q, or M-Q. Routines SORBDB1, SORBDB3, and SORBDB4 handle cases in which P is not the minimum dimension.

The orthogonal matrices P1, P2, and Q1 are P-by-P, (M-P)-by-(M-P), and (M-Q)-by-(M-Q), respectively. They are represented implicitly by Householder vectors.

B11 and B12 are P-by-P bidiagonal matrices represented implicitly by angles THETA, PHI.

Syntax

Fortran specification:

```
use armpl_library

subroutine sorbdb2(M, P, Q, X11, LDX11, X21, LDX21, THETA, PHI, TAUP1, TAUP2,
                   TAUQ1, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sorbdb2_(const armpl_int_t *m, const armpl_int_t *p,
              const armpl_int_t *q, float *x11, const armpl_int_t *ldx11,
              float *x21, const armpl_int_t *ldx21, float *theta, float *phi,
              float *taup1, float *taup2, float *tauq1, float *work,
              const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows X11 plus the number of rows in X21.

P Input parameter.

P is INTEGER

The number of rows in X11. 0 <= P <= min(M-P, Q,M-Q).

Q Input parameter.

Q is INTEGER

The number of columns in X11 and X21. 0 <= Q <= M.

X11 Input and output parameter.

X11 is REAL

X11 is an array, dimension (LDX11, Q). On entry, the top block of the matrix X to be reduced. On exit, the columns of tril(X11) specify reflectors for P1 and the rows of triu(X11,1) specify reflectors for Q1.

LDX11 Input parameter.

LDX11 is INTEGER

The leading dimension of X11. LDX11 >= P.

X21 Input and output parameter.

X21 is REAL

X21 is an array, dimension (LDX21, Q). On entry, the bottom block of the matrix X to be reduced. On exit, the columns of tril(X21) specify reflectors for P2.

LDX21 Input parameter.

LDX21 is INTEGER

The leading dimension of X21. LDX21 >= M-P.

THETA Output parameter.

THETA is REAL

THETA is an array, dimension (Q). The entries of the bidiagonal blocks B11, B21 are defined by THETA and PHI. See Further Details.

PHI Output parameter.

PHI is REAL

PHI is an array, dimension (Q-1). The entries of the bidiagonal blocks B11, B21 are defined by THETA and PHI. See Further Details.

TAUP1 Output parameter.

TAUP1 is REAL

TAUP1 is an array, dimension (P). The scalar factors of the elementary reflectors that define P1.

TAUP2 Output parameter.

TAUP2 is REAL

TAUP2 is an array, dimension (M-P). The scalar factors of the elementary reflectors that define P2.

TAUQ1 Output parameter.

TAUQ1 is REAL

TAUQ1 is an array, dimension (Q). The scalar factors of the elementary reflectors that define Q1.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= M-Q.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *dorbdb2*.

4.17.589 sorbdb3

sorbdb3 simultaneously bidiagonalizes the blocks of a tall and skinny matrix X with orthonomal columns:

```
                         [ B11 ]
[ X11 ]   [ P1 |    ] [  0  ]
[-----] = [---------] [-----] Q1**T .
[ X21 ]   [    | P2 ] [ B21 ]
                         [  0  ]
```

X11 is P-by-Q, and X21 is (M-P)-by-Q. M-P must be no larger than P, Q, or M-Q. Routines SORBDB1, SORBDB2, and SORBDB4 handle cases in which M-P is not the minimum dimension.

The orthogonal matrices P1, P2, and Q1 are P-by-P, (M-P)-by-(M-P), and (M-Q)-by-(M-Q), respectively. They are represented implicitly by Householder vectors.

B11 and B12 are (M-P)-by-(M-P) bidiagonal matrices represented implicitly by angles THETA, PHI.

Syntax

Fortran specification:

```
use armpl_library

subroutine sorbdb3(M, P, Q, X11, LDX11, X21, LDX21, THETA, PHI, TAUP1, TAUP2,
                   TAUQ1, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sorbdb3_(const armpl_int_t *m, const armpl_int_t *p,
              const armpl_int_t *q, float *x11, const armpl_int_t *ldx11,
              float *x21, const armpl_int_t *ldx21, float *theta, float *phi,
              float *taup1, float *taup2, float *tauq1, float *work,
              const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows X11 plus the number of rows in X21.

P Input parameter.

P is INTEGER

The number of rows in X11. 0 <= P <= M. M-P <= min(P, Q,M-Q).

Q Input parameter.

Q is INTEGER

The number of columns in X11 and X21. 0 <= Q <= M.

X11 Input and output parameter.

X11 is REAL

X11 is an array, dimension (LDX11, Q). On entry, the top block of the matrix X to be reduced. On exit, the columns of tril(X11) specify reflectors for P1 and the rows of triu(X11,1) specify reflectors for Q1.

LDX11 Input parameter.

LDX11 is INTEGER

The leading dimension of X11. LDX11 >= P.

X21 Input and output parameter.

X21 is REAL

X21 is an array, dimension (LDX21, Q). On entry, the bottom block of the matrix X to be reduced. On exit, the columns of tril(X21) specify reflectors for P2.

LDX21 Input parameter.

LDX21 is INTEGER

The leading dimension of X21. LDX21 >= M-P.

THETA Output parameter.

THETA is REAL

THETA is an array, dimension (Q). The entries of the bidiagonal blocks B11, B21 are defined by THETA and PHI. See Further Details.

PHI Output parameter.

PHI is REAL

PHI is an array, dimension (Q-1). The entries of the bidiagonal blocks B11, B21 are defined by THETA and PHI. See Further Details.

TAUP1 Output parameter.

TAUP1 is REAL

TAUP1 is an array, dimension (P). The scalar factors of the elementary reflectors that define P1.

TAUP2 Output parameter.

TAUP2 is REAL

TAUP2 is an array, dimension (M-P). The scalar factors of the elementary reflectors that define P2.

TAUQ1 Output parameter.

TAUQ1 is REAL

TAUQ1 is an array, dimension (Q). The scalar factors of the elementary reflectors that define Q1.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= M-Q.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *dorbdb3*.

4.17.590 sorbdb4

sorbdb4 simultaneously bidiagonalizes the blocks of a tall and skinny matrix X with orthonomal columns:

```
                         [ B11 ]
[ X11 ]   [ P1 |    ] [  0  ]
[-----] = [---------] [-----] Q1**T .
[ X21 ]   [    | P2 ] [ B21 ]
                         [  0  ]
```

X11 is P-by-Q, and X21 is (M-P)-by-Q. M-Q must be no larger than P, M-P, or Q. Routines SORBDB1, SORBDB2, and SORBDB3 handle cases in which M-Q is not the minimum dimension.

The orthogonal matrices P1, P2, and Q1 are P-by-P, (M-P)-by-(M-P), and (M-Q)-by-(M-Q), respectively. They are represented implicitly by Householder vectors.

B11 and B12 are (M-Q)-by-(M-Q) bidiagonal matrices represented implicitly by angles THETA, PHI.

Syntax

Fortran specification:

```
use armpl_library

subroutine sorbdb4(M, P, Q, X11, LDX11, X21, LDX21, THETA, PHI, TAUP1, TAUP2,
                   TAUQ1, PHANTOM, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sorbdb4_(const armpl_int_t *m, const armpl_int_t *p,
              const armpl_int_t *q, float *x11, const armpl_int_t *ldx11,
              float *x21, const armpl_int_t *ldx21, float *theta, float *phi,
              float *taup1, float *taup2, float *tauq1, float *phantom,
              float *work, const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows X11 plus the number of rows in X21.

P Input parameter.

P is INTEGER

The number of rows in X11. 0 <= P <= M.

Q Input parameter.

Q is INTEGER

The number of columns in X11 and X21. 0 <= Q <= M and M-Q <= min(P,M-P, Q).

X11 Input and output parameter.

X11 is REAL

X11 is an array, dimension (LDX11, Q). On entry, the top block of the matrix X to be reduced. On exit, the columns of tril(X11) specify reflectors for P1 and the rows of triu(X11,1) specify reflectors for Q1.

LDX11 Input parameter.

LDX11 is INTEGER

The leading dimension of X11. LDX11 >= P.

X21 Input and output parameter.

X21 is REAL

X21 is an array, dimension (LDX21, Q). On entry, the bottom block of the matrix X to be reduced. On exit, the columns of tril(X21) specify reflectors for P2.

LDX21 Input parameter.

LDX21 is INTEGER

The leading dimension of X21. LDX21 >= M-P.

THETA Output parameter.

THETA is REAL

THETA is an array, dimension (Q). The entries of the bidiagonal blocks B11, B21 are defined by THETA and PHI. See Further Details.

PHI Output parameter.

PHI is REAL

PHI is an array, dimension (Q-1). The entries of the bidiagonal blocks B11, B21 are defined by THETA and PHI. See Further Details.

TAUP1 Output parameter.

TAUP1 is REAL

TAUP1 is an array, dimension (P). The scalar factors of the elementary reflectors that define P1.

TAUP2 Output parameter.

TAUP2 is REAL

TAUP2 is an array, dimension (M-P). The scalar factors of the elementary reflectors that define P2.

TAUQ1 Output parameter.

TAUQ1 is REAL

TAUQ1 is an array, dimension (Q). The scalar factors of the elementary reflectors that define Q1.

PHANTOM Output parameter.

PHANTOM is REAL

PHANTOM is an array, dimension (M). The routine computes an M-by-1 column vector Y that is orthogonal to the columns of [X11; X21]. PHANTOM(1:P) and PHANTOM(P+1:M) contain Householder vectors for Y(1:P) and Y(P+1:M), respectively.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= M-Q.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *dorbdb4*.

4.17.591 sorbdb5

sorbdb5 orthogonalizes the column vector

```
X = [ X1 ]
    [ X2 ]
```

with respect to the columns of

```
Q = [ Q1 ] .
    [ Q2 ]
```

The columns of Q must be orthonormal.

If the projection is zero according to Kahan's "twice is enough" criterion, then some other vector from the orthogonal complement is returned. This vector is chosen in an arbitrary but deterministic way.

Syntax

Fortran specification:

```
use armpl_library

subroutine sorbdb5(M1, M2, N, X1, INCX1, X2, INCX2, Q1, LDQ1, Q2, LDQ2, WORK,
                   LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sorbdb5_(const armpl_int_t *m1, const armpl_int_t *m2,
              const armpl_int_t *n, float *x1, const armpl_int_t *incx1,
              float *x2, const armpl_int_t *incx2, float *q1,
              const armpl_int_t *ldq1, float *q2, const armpl_int_t *ldq2,
              float *work, const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M1 Input parameter.

M1 is INTEGER

The dimension of X1 and the number of rows in Q1. 0 <= M1.

M2 Input parameter.

M2 is INTEGER

The dimension of X2 and the number of rows in Q2. 0 <= M2.

N Input parameter.

N is INTEGER

The number of columns in Q1 and Q2. 0 <= N.

X1 Input and output parameter.

X1 is REAL

X1 is an array, dimension (M1). On entry, the top part of the vector to be orthogonalized. On exit, the top part of the projected vector.

INCX1 Input parameter.

INCX1 is INTEGER

Increment for entries of X1.

X2 Input and output parameter.

X2 is REAL

X2 is an array, dimension (M2). On entry, the bottom part of the vector to be orthogonalized. On exit, the bottom part of the projected vector.

INCX2 Input parameter.

INCX2 is INTEGER

Increment for entries of X2.

Q1 Input parameter.

Q1 is REAL

Q1 is an array, dimension (LDQ1, N). The top part of the orthonormal basis matrix.

LDQ1 Input parameter.

LDQ1 is INTEGER

The leading dimension of Q1. LDQ1 >= M1.

Q2 Input parameter.

Q2 is REAL

Q2 is an array, dimension (LDQ2, N). The bottom part of the orthonormal basis matrix.

LDQ2 Input parameter.

LDQ2 is INTEGER

The leading dimension of Q2. LDQ2 >= M2.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= N.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *dorbdb5*.

4.17.592 sorbdb6

sorbdb6 orthogonalizes the column vector

```
X = [ X1 ]
    [ X2 ]
```

with respect to the columns of

```
Q = [ Q1 ] .
    [ Q2 ]
```

The columns of Q must be orthonormal.

If the projection is zero according to Kahan's "twice is enough" criterion, then the zero vector is returned.

Syntax

Fortran specification:

```
use armpl_library

subroutine sorbdb6(M1, M2, N, X1, INCX1, X2, INCX2, Q1, LDQ1, Q2, LDQ2, WORK,
                   LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void sorbdb6_(const armpl_int_t *m1, const armpl_int_t *m2,
              const armpl_int_t *n, float *x1, const armpl_int_t *incx1,
              float *x2, const armpl_int_t *incx2, float *q1,
              const armpl_int_t *ldq1, float *q2, const armpl_int_t *ldq2,
              float *work, const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M1 Input parameter.

M1 is INTEGER

The dimension of X1 and the number of rows in Q1. 0 <= M1.

M2 Input parameter.

M2 is INTEGER

The dimension of X2 and the number of rows in Q2. 0 <= M2.

N Input parameter.

N is INTEGER

The number of columns in Q1 and Q2. 0 <= N.

X1 Input and output parameter.

X1 is REAL

X1 is an array, dimension (M1). On entry, the top part of the vector to be orthogonalized. On exit, the top part of the projected vector.

INCX1 Input parameter.

INCX1 is INTEGER

Increment for entries of X1.

X2 Input and output parameter.

X2 is REAL

X2 is an array, dimension (M2). On entry, the bottom part of the vector to be orthogonalized. On exit, the bottom part of the projected vector.

INCX2 Input parameter.

INCX2 is INTEGER

Increment for entries of X2.

Q1 Input parameter.

Q1 is REAL

Q1 is an array, dimension (LDQ1, N). The top part of the orthonormal basis matrix.

LDQ1 Input parameter.

LDQ1 is INTEGER

The leading dimension of Q1. LDQ1 >= M1.

Q2 Input parameter.

Q2 is REAL

Q2 is an array, dimension (LDQ2, N). The bottom part of the orthonormal basis matrix.

LDQ2 Input parameter.

LDQ2 is INTEGER

The leading dimension of Q2. LDQ2 >= M2.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= N.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *dorbdb6*.

4.17.593 sorg2l

sorg2l generates an m by n real matrix Q with orthonormal columns, which is defined as the last n columns of a product of k elementary reflectors of order m

```
Q  =  H(k) . . . H(2) H(1)
```

as returned by SGEQLF.

Syntax

Fortran specification:

```
use armpl_library

subroutine sorg2l(M, N, K, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void sorg2l_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *k,
             float *a, const armpl_int_t *lda, const float *tau, float *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix Q. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix Q. M >= N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. N >= K >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the (n-k+i)-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by SGEQLF in the last k columns of its array argument A. On exit, the m by n matrix Q.

LDA Input parameter.

LDA is INTEGER

The first dimension of the array A. LDA >= max(1, M).

TAU Input parameter.

TAU is REAL

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by SGEQLF.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument has an illegal value

Related Information

For this routine in other precisions, please see *dorg2l*.

4.17.594 sorg2r

sorg2r generates an m by n real matrix Q with orthonormal columns, which is defined as the first n columns of a product of k elementary reflectors of order m

```
Q  =  H(1) H(2) . . . H(k)
```

as returned by SGEQRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine sorg2r(M, N, K, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void sorg2r_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *k,
             float *a, const armpl_int_t *lda, const float *tau, float *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix Q. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix Q. M >= N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. N >= K >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the i-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by SGEQRF in the first k columns of its array argument A. On exit, the m-by-n matrix Q.

LDA Input parameter.

LDA is INTEGER

The first dimension of the array A. LDA >= max(1, M).

TAU Input parameter.

TAU is REAL

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by SGEQRF.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument has an illegal value

Related Information

For this routine in other precisions, please see *dorg2r*.

4.17.595 sorgl2

`sorgl2` generates an m by n real matrix Q with orthonormal rows, which is defined as the first m rows of a product of k elementary reflectors of order n

```
Q  =  H(k) . . . H(2) H(1)
```

as returned by SGELQF.

Syntax

Fortran specification:

```
use armpl_library

subroutine sorgl2(M, N, K, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void sorgl2_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *k,
             float *a, const armpl_int_t *lda, const float *tau, float *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

`M` is `INTEGER`

The number of rows of the matrix Q. `M` >= 0.

N Input parameter.

`N` is `INTEGER`

The number of columns of the matrix Q. `N` >= `M`.

K Input parameter.

`K` is `INTEGER`

The number of elementary reflectors whose product defines the matrix Q. `M` >= `K` >= 0.

A Input and output parameter.

`A` is `REAL`

`A` is an array, dimension (`LDA`, `N`). On entry, the i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by SGELQF in the first k rows of its array argument `A`. On exit, the m-by-n matrix Q.

LDA Input parameter.

`LDA` is `INTEGER`

The first dimension of the array `A`. `LDA` >= max(1, `M`).

TAU Input parameter.

`TAU` is `REAL`

`TAU` is an array, dimension (`K`). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by SGELQF.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (M) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument has an illegal value

Related Information

For this routine in other precisions, please see *dorgl2*.

4.17.596 sorgr2

sorgr2 generates an m by n real matrix Q with orthonormal rows, which is defined as the last m rows of a product of k elementary reflectors of order n

```
Q  =  H(1) H(2) . . . H(k)
```

as returned by SGERQF.

Syntax

Fortran specification:

```
use armpl_library

subroutine sorgr2(M, N, K, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void sorgr2_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *k,
             float *a, const armpl_int_t *lda, const float *tau, float *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix Q. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix Q. N >= M.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. M >= K >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the (m-k+i)-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by SGERQF in the last k rows of its array argument A. On exit, the m by n matrix Q.

LDA Input parameter.

LDA is INTEGER

The first dimension of the array A. LDA >= max(1, M).

TAU Input parameter.

TAU is REAL

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by SGERQF.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (M) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument has an illegal value

Related Information

For this routine in other precisions, please see *dorgr2*.

4.17.597 sorm2l

sorm2l overwrites the general real m by n matrix C with

```
Q * C  if SIDE = 'L' and TRANS = 'N', or

Q**T * C  if SIDE = 'L' and TRANS = 'T', or

C * Q  if SIDE = 'R' and TRANS = 'N', or

C * Q**T if SIDE = 'R' and TRANS = 'T',
```

where Q is a real orthogonal matrix defined as the product of k elementary reflectors

```
Q = H(k) . . . H(2) H(1)
```

as returned by SGEQLF. Q is of order m if SIDE = 'L' and of order n if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine sorm2l(SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void sorm2l_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k, const float *a,
             const armpl_int_t *lda, const float *tau, float *c,
             const armpl_int_t *ldc, float *work, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left = 'R': apply Q or Q^T from the Right

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply Q (No transpose) = 'T': apply Q^T (Transpose)

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, K). The i-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by SGEQLF in the last k columns of its array argument A. A is modified by the routine but restored on exit.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDA >= max(1, M); if SIDE = 'R', LDA >= max(1, N).

TAU Input parameter.

TAU is REAL

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by SGEQLF.

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, N). On entry, the m by n matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension. (N) if SIDE = 'L', (M) if SIDE = 'R'

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dorm2l*.

4.17.598 sorm2r

sorm2r overwrites the general real m by n matrix C with

```
Q * C  if SIDE = 'L' and TRANS = 'N', or

Q**T* C  if SIDE = 'L' and TRANS = 'T', or

C * Q  if SIDE = 'R' and TRANS = 'N', or

C * Q**T if SIDE = 'R' and TRANS = 'T',
```

where Q is a real orthogonal matrix defined as the product of k elementary reflectors

```
Q = H(1) H(2) . . . H(k)
```

as returned by SGEQRF. Q is of order m if SIDE = 'L' and of order n if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine sorm2r(SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void sorm2r_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k, const float *a,
             const armpl_int_t *lda, const float *tau, float *c,
             const armpl_int_t *ldc, float *work, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left = 'R': apply Q or Q^T from the Right

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply Q (No transpose) = 'T': apply Q^T (Transpose)

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, K). The i-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,. . . ,k, as returned by SGEQRF in the first k columns of its array argument A. A is modified by the routine but restored on exit.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDA >= max(1, M); if SIDE = 'R', LDA >= max(1, N).

TAU Input parameter.

TAU is REAL

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by SGEQRF.

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, N). On entry, the m by n matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension. (N) if SIDE = 'L', (M) if SIDE = 'R'

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dorm2r*.

4.17.599 sorml2

sorml2 overwrites the general real m by n matrix C with

```
Q * C  if SIDE = 'L' and TRANS = 'N', or

Q**T* C  if SIDE = 'L' and TRANS = 'T', or

C * Q  if SIDE = 'R' and TRANS = 'N', or

C * Q**T if SIDE = 'R' and TRANS = 'T',
```

where Q is a real orthogonal matrix defined as the product of k elementary reflectors

```
Q = H(k) . . . H(2) H(1)
```

as returned by SGELQF. Q is of order m if SIDE = 'L' and of order n if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine sorml2(SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void sorml2_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k, const float *a,
             const armpl_int_t *lda, const float *tau, float *c,
             const armpl_int_t *ldc, float *work, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left = 'R': apply Q or Q^T from the Right

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply Q (No transpose) = 'T': apply Q^T (Transpose)

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is REAL

A is an array, dimension. (LDA, M) if SIDE = 'L', (LDA, N) if SIDE = 'R' The i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by SGELQF in the first k rows of its array argument A. A is modified by the routine but restored on exit.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, K).

TAU Input parameter.

TAU is REAL

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by SGELQF.

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, N). On entry, the m by n matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension. (N) if SIDE = 'L', (M) if SIDE = 'R'

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dorml2*.

4.17.600 sormr2

sormr2 overwrites the general real m by n matrix C with

```
Q * C  if SIDE = 'L' and TRANS = 'N', or

Q**T* C  if SIDE = 'L' and TRANS = 'T', or

C * Q  if SIDE = 'R' and TRANS = 'N', or

C * Q**T if SIDE = 'R' and TRANS = 'T',
```

where Q is a real orthogonal matrix defined as the product of k elementary reflectors

```
Q = H(1) H(2) . . . H(k)
```

as returned by SGERQF. Q is of order m if SIDE = 'L' and of order n if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine sormr2(SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void sormr2_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k, const float *a,
             const armpl_int_t *lda, const float *tau, float *c,
             const armpl_int_t *ldc, float *work, armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left = 'R': apply Q or Q^T from the Right

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply Q (No transpose) = 'T': apply Q' (Transpose)

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is REAL

A is an array, dimension. (LDA, M) if SIDE = 'L', (LDA, N) if SIDE = 'R' The i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by SGERQF in the last k rows of its array argument A. A is modified by the routine but restored on exit.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, K).

TAU Input parameter.

TAU is REAL

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by SGERQF.

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, N). On entry, the m by n matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension. (N) if SIDE = 'L', (M) if SIDE = 'R'

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dormr2*.

4.17.601 sormr3

sormr3 overwrites the general real m by n matrix C with

```
Q * C  if SIDE = 'L' and TRANS = 'N', or

Q**T* C  if SIDE = 'L' and TRANS = 'C', or

C * Q  if SIDE = 'R' and TRANS = 'N', or

C * Q**T if SIDE = 'R' and TRANS = 'C',
```

where Q is a real orthogonal matrix defined as the product of k elementary reflectors

```
Q = H(1) H(2) . . . H(k)
```

as returned by STZRZF. Q is of order m if SIDE = 'L' and of order n if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine sormr3(SIDE, TRANS, M, N, K, L, A, LDA, TAU, C, LDC, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void sormr3_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k, const armpl_int_t *l,
             const float *a, const armpl_int_t *lda, const float *tau,
             float *c, const armpl_int_t *ldc, float *work, armpl_int_t *info,
             ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^T from the Left = 'R': apply Q or Q^T from the Right

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply Q (No transpose) = 'T': apply Q^T (Transpose)

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

L Input parameter.

L is INTEGER

The number of columns of the matrix A containing the meaningful part of the Householder reflectors. If SIDE = 'L', M >= L >= 0, if SIDE = 'R', N >= L >= 0.

A Input parameter.

A is REAL

A is an array, dimension. (LDA, M) if SIDE = 'L', (LDA, N) if SIDE = 'R' The i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,. . . ,k, as returned by STZRZF in the last k rows of its array argument A. A is modified by the routine but restored on exit.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, K).

TAU Input parameter.

TAU is REAL

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by STZRZF.

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, N). On entry, the m-by-n matrix C. On exit, C is overwritten by Q* C or Q^T * C or C*Q^T or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension. (N) if SIDE = 'L', (M) if SIDE = 'R'

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *dormr3*.

4.17.602 spbtf2

spbtf2 computes the Cholesky factorization of a real symmetric positive definite band matrix A.

The factorization has the form

```
A = U**T * U ,  if UPLO = 'U', or
A = L  * L**T,  if UPLO = 'L',
```

where U is an upper triangular matrix, U^T is the transpose of U, and L is lower triangular.

This is the unblocked version of the algorithm, calling Level 2 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine spbtf2(UPLO, N, KD, AB, LDAB, INFO)
```

C specification:

```
#include "armpl.h"

void spbtf2_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
             float *ab, const armpl_int_t *ldab, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of super-diagonals of the matrix A if UPLO = 'U', or the number of sub-diagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is REAL

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, if INFO = 0, the triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T of the band matrix A, in the same storage format as A.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value > 0: if INFO = k, the leading minor of order k is not positive definite, and the factorization could not be completed.

Related Information

For this routine in other precisions, please see *cpbtf2*, *dpbtf2* and *zpbtf2*.

4.17.603 spotf2

spotf2 computes the Cholesky factorization of a real symmetric positive definite matrix A.

The factorization has the form

```
A = U**T * U ,  if UPLO = 'U', or
A = L  * L**T,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is lower triangular.

This is the unblocked version of the algorithm, calling Level 2 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine spotf2(UPLO, N, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void spotf2_(const char *uplo, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored. = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading n by n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n by n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T .

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value > 0: if INFO = k, the leading minor of order k is not positive definite, and the factorization could not be completed.

Related Information

For this routine in other precisions, please see *cpotf2*, *dpotf2* and *zpotf2*.

4.17.604 spstf2

spstf2 computes the Cholesky factorization with complete pivoting of a real symmetric positive semidefinite matrix A.

The factorization has the form

```
P**T * A * P = U**T * U ,  if UPLO = 'U',
P**T * A * P = L  * L**T,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is lower triangular, and P is stored as vector PIV.

This algorithm does not attempt to check that A is positive semidefinite. This version of the algorithm calls level 2 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine spstf2(UPLO, N, A, LDA, PIV, RANK, TOL, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void spstf2_(const char *uplo, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, armpl_int_t *piv, armpl_int_t *rank,
             const float *tol, float *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored. = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading n by n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n by n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization as above.

PIV Output parameter.

PIV is INTEGER array, dimension (N)

PIV is such that the nonzero entries are P(PIV(K), K) = 1.

RANK Output parameter.

RANK is INTEGER

The rank of A given by the number of steps the algorithm completed.

TOL Input parameter.

TOL is REAL

User defined tolerance. If TOL < 0, then N*U*MAX(A(K,K)) will be used. The algorithm terminates at the (K-1)st step if the pivot <= TOL.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (2*N). Work space.

INFO Output parameter.

INFO is INTEGER

< 0: If INFO = -K, the K-th argument had an illegal value, = 0: algorithm completed successfully, and > 0: the matrix A is either rank deficient with computed rank as returned in RANK, or is not positive semidefinite. See Section 7 of LAPACK Working Note #161 for further information.

Related Information

For this routine in other precisions, please see *cpstf2*, *dpstf2* and *zpstf2*.

4.17.605 sptts2

sptts2 solves a tridiagonal system of the form

```
A * X = B
```

using the L*D*L^T factorization of A computed by SPTTRF. D is a diagonal matrix specified in the vector D, L is a unit bidiagonal matrix whose subdiagonal is specified in the vector E, and X and B are N by NRHS matrices.

Syntax

Fortran specification:

```
use armpl_library

subroutine sptts2(N, NRHS, D, E, B, LDB)
```

C specification:

```
#include "armpl.h"

void sptts2_(const armpl_int_t *n, const armpl_int_t *nrhs, const float *d,
             const float *e, float *b, const armpl_int_t *ldb);
```

Parameters

N Input parameter.

N is INTEGER

The order of the tridiagonal matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

D Input parameter.

D is REAL

D is an array, dimension (N). The n diagonal elements of the diagonal matrix D from the L*D*L^T factorization of A.

E Input parameter.

E is REAL

E is an array, dimension (N-1). The (n-1) subdiagonal elements of the unit bidiagonal factor L from the L*D*L^T factorization of A. E can also be regarded as the superdiagonal of the unit bidiagonal factor U from the factorization A = U^T *D*U.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, NRHS). On entry, the right hand side vectors B for the system of linear equations. On exit, the solution vectors, X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Related Information

For this routine in other precisions, please see *cptts2*, *dptts2* and *zptts2*.

4.17.606 srscl

srscl multiplies an n-element real vector x by the real scalar 1/a. This is done without overflow or underflow as long as the final result x/a does not overflow or underflow.

Syntax

Fortran specification:

```
use armpl_library

subroutine srscl(N, SA, SX, INCX)
```

C specification:

```
#include "armpl.h"

void srscl_(const armpl_int_t *n, const float *sa, float *sx,
            const armpl_int_t *incx);
```

Parameters

N Input parameter.

N is INTEGER

The number of components of the vector x.

SA Input parameter.

SA is REAL

The scalar a which is used to divide each component of x. SA must be >= 0, or the subroutine will divide by zero.

SX Input and output parameter.

SX is REAL

SX is an array, dimension. (1+(N-1)*abs(INCX)) The n-element vector x.

INCX Input parameter.

INCX is INTEGER

The increment between successive values of the vector SX. > 0: SX(1) = X(1) and SX(1+(i-1)*INCX) = x(i), 1< i<= n

Related Information

For this routine in other precisions, please see *drscl*.

4.17.607 ssfrk

Level 3 BLAS like routine for C in RFP Format.

ssfrk performs one of the symmetric rank–k operations

```
C := alpha*A*A**T + beta*C,
```

or

```
C := alpha*A**T*A + beta*C,
```

where alpha and beta are real scalars, C is an n–by–n symmetric matrix and A is an n–by–k matrix in the first case and a k–by–n matrix in the second case.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssfrk(TRANSR, UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C)
```

C specification:

```
#include "armpl.h"

void ssfrk_(const char *transr, const char *uplo, const char *trans,
            const armpl_int_t *n, const armpl_int_t *k, const float *alpha,
            const float *a, const armpl_int_t *lda, const float *beta,
            float *c, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': The Normal Form of RFP A is stored; = 'T': The Transpose Form of RFP A is stored.

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the array C is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of C is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of C is to be referenced.

Unchanged on exit.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' C := alpha*A*A^T + beta*C.

TRANS = 'T' or 't' C := alpha*A^T * A + beta*C.

Unchanged on exit.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix C. N must be at least zero. Unchanged on exit.

K Input parameter.

K is INTEGER

On entry with TRANS = 'N' or 'n', K specifies the number of columns of the matrix A, and on entry with TRANS = 'T' or 't', K specifies the number of rows of the matrix A. K must be at least zero. Unchanged on exit.

ALPHA Input parameter.

ALPHA is REAL

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

A Input parameter.

A is REAL

A is an array, dimension (LDA,ka). where KA is K when TRANS = 'N' or 'n', and is N otherwise. Before entry with TRANS = 'N' or 'n', the leading N–by–K part of the array A must contain the matrix A, otherwise the leading K–by–N part of the array A must contain the matrix A. Unchanged on exit.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDA must be at least max(1, n), otherwise LDA must be at least max(1, k). Unchanged on exit.

BETA Input parameter.

BETA is REAL

On entry, BETA specifies the scalar beta. Unchanged on exit.

C Input and output parameter.

C is REAL

C is an array, dimension (NT). NT = N*(N+1)/2. On entry, the symmetric matrix C in RFP Format. RFP Format is described by TRANSR, UPLO and N.

Related Information

For this routine in other precisions, please see *dsfrk*. It also exists with a native C interface as *LAPACKE_ssfrk*.

4.17.608 ssyconv

ssyconv convert A given by TRF into L and D and vice-versa. Get Non-diag elements of D (returned in workspace) and apply or reverse permutation done in TRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssyconv(UPLO, WAY, N, A, LDA, IPIV, E, INFO)
```

C specification:

```
#include "armpl.h"

void ssyconv_(const char *uplo, const char *way, const armpl_int_t *n,
              float *a, const armpl_int_t *lda, const armpl_int_t *ipiv,
              float *e, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

WAY Input parameter.

WAY is CHARACTER*1

= 'C': Convert = 'R': Revert

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by SSYTRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by SSYTRF.

E Output parameter.

E is REAL

E is an array, dimension (N). E stores the supdiagonal/subdiagonal of the symmetric 1-by-1 or 2-by-2 block diagonal matrix D in LDLT.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csyconv*, *dsyconv* and *zsyconv*. It also exists with a native C interface as *LAPACKE_ssyconv*.

4.17.609 ssygs2

ssygs2 reduces a real symmetric-definite generalized eigenproblem to standard form.

If ITYPE = 1, the problem is A*x = lambda*B*x, and A is overwritten by inv(U^T)*A*inv(U) or inv(L)*A*inv(L^T)

If ITYPE = 2 or 3, the problem is A*B*x = lambda*x or B*A*x = lambda*x, and A is overwritten by U*A*U^T or L^T*A*L.

B must have been previously factorized as U^T*U or L*L^T by SPOTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssygs2(ITYPE, UPLO, N, A, LDA, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void ssygs2_(const armpl_int_t *itype, const char *uplo, const armpl_int_t *n,
             float *a, const armpl_int_t *lda, const float *b,
             const armpl_int_t *ldb, armpl_int_t *info, ... );
```

Parameters

ITYPE Input parameter.

ITYPE is INTEGER

= 1: compute inv(U^T)*A*inv(U) or inv(L)*A*inv(L^T); = 2 or 3: compute U*A*U^T or L^T*A*L.

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored, and how B has been factorized. = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading n by n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n by n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the transformed matrix, stored in the same format as A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input parameter.

B is REAL

B is an array, dimension (LDB, N). The triangular factor from the Cholesky factorization of B, as returned by SPOTRF.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *dsygs2*.

4.17.610 ssyswapr

SSYSWAPR applies an elementary permutation on the rows and the columns of a symmetric matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssyswapr(UPLO, N, A, LDA, I1, I2)
```

C specification:

```
#include "armpl.h"

void ssyswapr_(const char *uplo, const armpl_int_t *n, float *a,
               const armpl_int_t *lda, const armpl_int_t *i1,
               const armpl_int_t *i2, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the NB diagonal matrix D and the multipliers used to obtain the factor U or L as computed by SSYTRF.

On exit, if INFO = 0, the (symmetric) inverse of the original matrix. If UPLO = 'U', the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; if UPLO = 'L' the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

I1 Input parameter.

I1 is INTEGER

Index of the first row to swap

I2 Input parameter.

I2 is INTEGER

Index of the second row to swap

Related Information

For this routine in other precisions, please see *csyswapr*, *dsyswapr* and *zsyswapr*. It also exists with a native C interface as *LAPACKE_ssyswapr*.

4.17.611 ssytd2

ssytd2 reduces a real symmetric matrix A to symmetric tridiagonal form T by an orthogonal similarity transformation: Q^T * A * Q = T.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssytd2(UPLO, N, A, LDA, D, E, TAU, INFO)
```

C specification:

```
#include "armpl.h"

void ssytd2_(const char *uplo, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, float *d, float *e, float *tau,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, if UPLO = 'U', the diagonal and first superdiagonal of A are overwritten by the corresponding elements of the tridiagonal matrix T, and the elements above the first superdiagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors; if UPLO = 'L', the diagonal and first subdiagonal of A are over- written by the corresponding elements of the tridiagonal matrix T, and the elements below the first subdiagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

D Output parameter.

D is REAL

D is an array, dimension (N). The diagonal elements of the tridiagonal matrix T: D(i) = A(i,i).

E Output parameter.

E is REAL

E is an array, dimension (N-1). The off-diagonal elements of the tridiagonal matrix T: E(i) = A(i,i+1) if UPLO = 'U', E(i) = A(i+1,i) if UPLO = 'L'.

TAU Output parameter.

TAU is REAL

TAU is an array, dimension (N-1). The scalar factors of the elementary reflectors (see Further Details).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *dsytd2*.

4.17.612 ssytf2

ssytf2 computes the factorization of a real symmetric matrix A using the Bunch-Kaufman diagonal pivoting method:

```
A = U*D*U**T  or  A = L*D*L**T
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, U^T is the transpose of U, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the unblocked version of the algorithm, calling Level 2 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssytf2(UPLO, N, A, LDA, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void ssytf2_(const char *uplo, const armpl_int_t *n, float *a,
             const armpl_int_t *lda, armpl_int_t *ipiv, armpl_int_t *info,
             ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of

A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L (see below for further details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D.

If UPLO = 'U': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block.

If UPLO = 'L': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value > 0: if INFO = k, D(k,k) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *csytf2*, *dsytf2* and *zsytf2*.

4.17.613 ssytf2_rook

SSYTF2_ROOK computes the factorization of a real symmetric matrix A using the bounded Bunch-Kaufman ("rook") diagonal pivoting method:

```
A = U*D*U**T  or  A = L*D*L**T
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, U^T is the transpose of U, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the unblocked version of the algorithm, calling Level 2 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine ssytf2_rook(UPLO, N, A, LDA, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void ssytf2_rook_(const char *uplo, const armpl_int_t *n, float *a,
                  const armpl_int_t *lda, armpl_int_t *ipiv,
                  armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L (see below for further details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D.

If UPLO = 'U': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k-1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k-1 and -IPIV(k-1) were inerchaged, D(k-1:k,k-1:k) is a 2-by-2 diagonal block.

If UPLO = 'L': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k+1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k+1 and -IPIV(k+1) were inerchaged, D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value > 0: if INFO = k, D(k,k) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *csytf2_rook*, *dsytf2_rook* and *zsytf2_rook*.

4.17.614 stfsm

Level 3 BLAS like routine for A in RFP Format.

`stfsm` solves the matrix equation

```
op( A )*X = alpha*B  or  X*op( A ) = alpha*B
```

where alpha is a scalar, X and B are m by n matrices, A is a unit, or non-unit, upper or lower triangular matrix and op(A) is one of

```
op( A ) = A   or   op( A ) = A**T.
```

A is in Rectangular Full Packed (RFP) Format.

The matrix X is overwritten on B.

Syntax

Fortran specification:

```
use armpl_library

subroutine stfsm(TRANSR, SIDE, UPLO, TRANS, DIAG, M, N, ALPHA, A, B, LDB)
```

C specification:

```
#include "armpl.h"

void stfsm_(const char *transr, const char *side, const char *uplo,
            const char *trans, const char *diag, const armpl_int_t *m,
            const armpl_int_t *n, const float *alpha, const float *a,
            float *b, const armpl_int_t *ldb, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': The Normal Form of RFP A is stored; = 'T': The Transpose Form of RFP A is stored.

SIDE Input parameter.

SIDE is CHARACTER*1

On entry, SIDE specifies whether op(A) appears on the left or right of X as follows:

SIDE = 'L' or 'l' op(A)*X = alpha*B.

SIDE = 'R' or 'r' X*op(A) = alpha*B.

Unchanged on exit.

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the RFP matrix A came from an upper or lower triangular matrix as follows: UPLO = 'U' or 'u' RFP A came from an upper triangular matrix UPLO = 'L' or 'l' RFP A came from a lower triangular matrix

Unchanged on exit.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the form of op(A) to be used in the matrix multiplication as follows:

TRANS = 'N' or 'n' op(A) = A.

TRANS = 'T' or 't' op(A) = A'.

Unchanged on exit.

DIAG Input parameter.

DIAG is CHARACTER*1

On entry, DIAG specifies whether or not RFP A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

Unchanged on exit.

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of B. M must be at least zero. Unchanged on exit.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of B. N must be at least zero. Unchanged on exit.

ALPHA Input parameter.

ALPHA is REAL

On entry, ALPHA specifies the scalar alpha. When alpha is zero then A is not referenced and B need not be set before entry. Unchanged on exit.

A Input parameter.

A is REAL

A is an array, dimension (NT). NT = N*(N+1)/2. On entry, the matrix A in RFP Format. RFP Format is described by TRANSR, UPLO and N as follows: If TRANSR='N' then RFP A is (0:N,0:K-1) when N is even; K=N/2. RFP A is (0:N-1,0:K) when N is odd; K=N/2. If TRANSR = 'T' then RFP is the transpose of RFP A as defined when TRANSR = 'N'. The contents of RFP A are defined by UPLO as follows: If UPLO = 'U' the RFP A contains the NT elements of upper packed A either in normal or transpose Format. If UPLO = 'L' the RFP A contains the NT elements of lower packed A either in normal or transpose Format. The LDA of RFP A is (N+1)/2 when TRANSR = 'T'. When TRANSR is 'N' the LDA is N+1 when N is even and is N when is odd. See the Note below for more details. Unchanged on exit.

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, N). Before entry, the leading m by n part of the array B must contain the right-hand side matrix B, and on exit is overwritten by the solution matrix X.

LDB Input parameter.

LDB is INTEGER

On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. LDB must be at least max(1, m). Unchanged on exit.

Related Information

For this routine in other precisions, please see *ctfsm*, *dtfsm* and *ztfsm*. It also exists with a native C interface as *LAPACKE_stfsm*.

4.17.615 stfttp

stfttp copies a triangular matrix A from rectangular full packed format (TF) to standard packed format (TP).

Syntax

Fortran specification:

```
use armpl_library

subroutine stfttp(TRANSR, UPLO, N, ARF, AP, INFO)
```

C specification:

```
#include "armpl.h"

void stfttp_(const char *transr, const char *uplo, const armpl_int_t *n,
             const float *arf, float *ap, armpl_int_t *info, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': ARF is in Normal format; = 'T': ARF is in Transpose format;

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

ARF Input parameter.

ARF is REAL

ARF is an array, dimension (N*(N+1)/2),. On entry, the upper or lower triangular matrix A stored in RFP format. For a further discussion see Notes below.

AP Output parameter.

AP is REAL

AP is an array, dimension (N*(N+1)/2),. On exit, the upper or lower triangular matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctfttp*, *dtfttp* and *ztfttp*. It also exists with a native C interface as *LAPACKE_stfttp*.

4.17.616 stfttr

stfttr copies a triangular matrix A from rectangular full packed format (TF) to standard full format (TR).

Syntax

Fortran specification:

```
use armpl_library

subroutine stfttr(TRANSR, UPLO, N, ARF, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void stfttr_(const char *transr, const char *uplo, const armpl_int_t *n,
             const float *arf, float *a, const armpl_int_t *lda,
             armpl_int_t *info, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': ARF is in Normal format; = 'T': ARF is in Transpose format.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

N Input parameter.

N is INTEGER

The order of the matrices ARF and A. N >= 0.

ARF Input parameter.

ARF is REAL

ARF is an array, dimension (N*(N+1)/2).. On entry, the upper (if UPLO = 'U') or lower (if UPLO = 'L') matrix A in RFP format. See the "Notes" below for more details.

A Output parameter.

A is REAL

A is an array, dimension (LDA, N). On exit, the triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is

not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctfttr*, *dtfttr* and *ztfttr*. It also exists with a native C interface as *LAPACKE_stfttr*.

4.17.617 stgex2

stgex2 swaps adjacent diagonal blocks (A11, B11) and (A22, B22) of size 1-by-1 or 2-by-2 in an upper (quasi) triangular matrix pair (A, B) by an orthogonal equivalence transformation.

(A, B) must be in generalized real Schur canonical form (as returned by SGGES), i.e. A is block upper triangular with 1-by-1 and 2-by-2 diagonal blocks. B is upper triangular.

Optionally, the matrices Q and Z of generalized Schur vectors are updated.

```
Q(in) * A(in) * Z(in)**T = Q(out) * A(out) * Z(out)**T
Q(in) * B(in) * Z(in)**T = Q(out) * B(out) * Z(out)**T
```

Syntax

Fortran specification:

```
use armpl_library

subroutine stgex2(WANTQ, WANTZ, N, A, LDA, B, LDB, Q, LDQ, Z, LDZ, J1, N1, N2,
                  WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void stgex2_(const armpl_int_t *wantq, const armpl_int_t *wantz,
             const armpl_int_t *n, float *a, const armpl_int_t *lda, float *b,
             const armpl_int_t *ldb, float *q, const armpl_int_t *ldq,
             float *z, const armpl_int_t *ldz, const armpl_int_t *j1,
             const armpl_int_t *n1, const armpl_int_t *n2, float *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

WANTQ Input parameter.

WANTQ is LOGICAL

.TRUE. : update the left transformation matrix Q; .FALSE.: do not update Q.

WANTZ Input parameter.

WANTZ is LOGICAL

.TRUE. : update the right transformation matrix Z; .FALSE.: do not update Z.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the matrix A in the pair (A, B). On exit, the updated matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, N). On entry, the matrix B in the pair (A, B). On exit, the updated matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Q Input and output parameter.

Q is REAL

Q is an array, dimension (LDQ, N). On entry, if WANTQ = .TRUE., the orthogonal matrix Q. On exit, the updated matrix Q. Not referenced if WANTQ = .FALSE..

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= 1. If WANTQ = .TRUE., LDQ >= N.

Z Input and output parameter.

Z is REAL

Z is an array, dimension (LDZ, N). On entry, if WANTZ =.TRUE., the orthogonal matrix Z. On exit, the updated matrix Z. Not referenced if WANTZ = .FALSE..

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1. If WANTZ = .TRUE., LDZ >= N.

J1 Input parameter.

J1 is INTEGER

The index to the first block (A11, B11). 1 <= J1 <= N.

N1 Input parameter.

N1 is INTEGER

The order of the first block (A11, B11). N1 = 0, 1 or 2.

N2 Input parameter.

N2 is INTEGER

The order of the second block (A22, B22). N2 = 0, 1 or 2.

WORK Output parameter.

WORK is REAL

WORK is an array, dimension (MAX(1, LWORK)). .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= MAX(N*(N2+N1), (N2+N1)*(N2+N1)*2)

INFO Output parameter.

INFO is INTEGER

=0: Successful exit >0: If INFO = 1, the transformed matrix (A, B) would be too far from generalized Schur form; the blocks are not swapped and (A, B) and (Q, Z) are unchanged. The problem of swapping is too ill-conditioned. <0: If INFO = -16: LWORK is too small. Appropriate value for LWORK is returned in WORK(1).

Related Information

For this routine in other precisions, please see *ctgex2*, *dtgex2* and *ztgex2*.

4.17.618 stgsy2

stgsy2 solves the generalized Sylvester equation:

```
A * R - L * B = scale * C                (1)
D * R - L * E = scale * F,
```

using Level 1 and 2 BLAS. where R and L are unknown M-by-N matrices, (A, D), (B, E) and (C, F) are given matrix pairs of size M-by-M, N-by-N and M-by-N, respectively, with real entries. (A, D) and (B, E) must be in generalized Schur canonical form, i.e. A, B are upper quasi triangular and D, E are upper triangular. The solution (R, L) overwrites (C, F). 0 <= SCALE <= 1 is an output scaling factor chosen to avoid overflow.

In matrix notation solving equation (1) corresponds to solve Z*x = scale*b, where Z is defined as

```
Z = [ kron(In, A)  -kron(B**T, Im) ]             (2)
    [ kron(In, D)  -kron(E**T, Im) ],
```

Ik is the identity matrix of size k and X^T is the transpose of X. kron(X, Y) is the Kronecker product between the matrices X and Y. In the process of solving (1), we solve a number of such systems where Dim(In), Dim(In) = 1 or 2.

If TRANS = 'T', solve the transposed system Z^T *y = scale*b for y, which is equivalent to solve for R and L in

```
A**T * R  + D**T * L   = scale * C           (3)
R  * B**T + L  * E**T  = scale * -F
```

This case is used to compute an estimate of Dif[(A, D), (B, E)] = sigma_min(Z) using reverse communicaton with SLACON.

stgsy2 also (IJOB >= 1) contributes to the computation in STGSYL of an upper bound on the separation between to matrix pairs. Then the input (A, D), (B, E) are sub-pencils of the matrix pair in STGSYL. See STGSYL for details.

Syntax

Fortran specification:

```
use armpl_library

subroutine stgsy2(TRANS, IJOB, M, N, A, LDA, B, LDB, C, LDC, D, LDD, E, LDE,
                  F, LDF, SCALE, RDSUM, RDSCAL, IWORK, PQ, INFO)
```

C specification:

```
#include "armpl.h"

void stgsy2_(const char *trans, const armpl_int_t *ijob, const armpl_int_t *m,
             const armpl_int_t *n, const float *a, const armpl_int_t *lda,
             const float *b, const armpl_int_t *ldb, float *c,
             const armpl_int_t *ldc, const float *d, const armpl_int_t *ldd,
             const float *e, const armpl_int_t *lde, float *f,
             const armpl_int_t *ldf, float *scale, float *rdsum,
             float *rdscal, armpl_int_t *iwork, armpl_int_t *pq,
             armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N', solve the generalized Sylvester equation (1). = 'T': solve the 'transposed' system (3).

IJOB Input parameter.

IJOB is INTEGER

Specifies what kind of functionality to be performed. = 0: solve (1) only. = 1: A contribution from this subsystem to a Frobenius norm-based estimate of the separation between two matrix pairs is computed. (look ahead strategy is used). = 2: A contribution from this subsystem to a Frobenius norm-based estimate of the separation between two matrix pairs is computed. (SGECON on sub-systems is used.) Not referenced if TRANS = 'T'.

M Input parameter.

M is INTEGER

On entry, M specifies the order of A and D, and the row dimension of C, F, R and L.

N Input parameter.

N is INTEGER

On entry, N specifies the order of B and E, and the column dimension of C, F, R and L.

A Input parameter.

A is REAL

A is an array, dimension (LDA, M). On entry, A contains an upper quasi triangular matrix.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the matrix A. LDA >= max(1, M).

B Input parameter.

B is REAL

B is an array, dimension (LDB, N). On entry, B contains an upper quasi triangular matrix.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the matrix B. LDB >= max(1, N).

C Input and output parameter.

C is REAL

C is an array, dimension (LDC, N). On entry, C contains the right-hand-side of the first matrix equation in (1). On exit, if IJOB = 0, C has been overwritten by the solution R.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the matrix C. LDC >= max(1, M).

D Input parameter.

D is REAL

D is an array, dimension (LDD, M). On entry, D contains an upper triangular matrix.

LDD Input parameter.

LDD is INTEGER

The leading dimension of the matrix D. LDD >= max(1, M).

E Input parameter.

E is REAL

E is an array, dimension (LDE, N). On entry, E contains an upper triangular matrix.

LDE Input parameter.

LDE is INTEGER

The leading dimension of the matrix E. LDE >= max(1, N).

F Input and output parameter.

F is REAL

F is an array, dimension (LDF, N). On entry, F contains the right-hand-side of the second matrix equation in (1). On exit, if IJOB = 0, F has been overwritten by the solution L.

LDF Input parameter.

LDF is INTEGER

The leading dimension of the matrix F. LDF >= max(1, M).

SCALE Output parameter.

SCALE is REAL

On exit, 0 <= SCALE <= 1. If 0 < SCALE < 1, the solutions R and L (C and F on entry) will hold the solutions to a slightly perturbed system but the input matrices A, B, D and E have not been changed. If SCALE = 0, R and L will hold the solutions to the homogeneous system with C = F = 0. Normally, SCALE = 1.

RDSUM Input and output parameter.

RDSUM is REAL

On entry, the sum of squares of computed contributions to the Dif-estimate under computation by STGSYL, where the scaling factor RDSCAL (see below) has been factored out. On exit, the corresponding sum of squares updated with the contributions from the current sub-system. If TRANS = 'T' RDSUM is not touched. NOTE: RDSUM only makes sense when STGSY2 is called by STGSYL.

RDSCAL Input and output parameter.

RDSCAL is REAL

On entry, scaling factor used to prevent overflow in RDSUM. On exit, RDSCAL is updated w.r.t. the current contributions in RDSUM. If TRANS = 'T', RDSCAL is not touched. NOTE: RDSCAL only makes sense when STGSY2 is called by STGSYL.

IWORK Output parameter.

IWORK is INTEGER array, dimension (M+N+2)

PQ Output parameter.

PQ is INTEGER

On exit, the number of subsystems (of size 2-by-2, 4-by-4 and 8-by-8) solved by this routine.

INFO Output parameter.

INFO is INTEGER

On exit, if INFO is set to =0: Successful exit <0: If INFO = -i, the i-th argument had an illegal value. >0: The matrix pairs (A, D) and (B, E) have common or very close eigenvalues.

Related Information

For this routine in other precisions, please see *ctgsy2*, *dtgsy2* and *ztgsy2*.

4.17.619 stplqt2

stplqt2 computes a LQ a factorization of a real "triangular-pentagonal" matrix C, which is composed of a triangular block A and pentagonal block B, using the compact WY representation for Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine stplqt2(M, N, L, A, LDA, B, LDB, T, LDT, INFO)
```

C specification:

```
#include "armpl.h"

void stplqt2_(const armpl_int_t *m, const armpl_int_t *n,
              const armpl_int_t *l, float *a, const armpl_int_t *lda,
              float *b, const armpl_int_t *ldb, float *t,
              const armpl_int_t *ldt, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The total number of rows of the matrix B. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix B, and the order of the triangular matrix A. N >= 0.

L Input parameter.

L is INTEGER

The number of rows of the lower trapezoidal part of B. MIN(M, N) >= L >= 0. See Further Details.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, M). On entry, the lower triangular M-by-M matrix A. On exit, the elements on and below the diagonal of the array contain the lower triangular matrix L.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, N). On entry, the pentagonal M-by-N matrix B. The first N-L columns are rectangular, and the last L columns are lower trapezoidal. On exit, B contains the pentagonal matrix V. See Further Details.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M).

T Output parameter.

T is REAL

T is an array, dimension (LDT, M). The N-by-N upper triangular factor T of the block reflector. See Further Details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, M)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctplqt2*, *dtplqt2* and *ztplqt2*.

4.17.620 stpqrt2

stpqrt2 computes a QR factorization of a real "triangular-pentagonal" matrix C, which is composed of a triangular block A and pentagonal block B, using the compact WY representation for Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine stpqrt2(M, N, L, A, LDA, B, LDB, T, LDT, INFO)
```

C specification:

```
#include "armpl.h"

void stpqrt2_(const armpl_int_t *m, const armpl_int_t *n,
              const armpl_int_t *l, float *a, const armpl_int_t *lda,
              float *b, const armpl_int_t *ldb, float *t,
              const armpl_int_t *ldt, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The total number of rows of the matrix B. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix B, and the order of the triangular matrix A. N >= 0.

L Input parameter.

L is INTEGER

The number of rows of the upper trapezoidal part of B. MIN(M, N) >= L >= 0. See Further Details.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the upper triangular N-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the upper triangular matrix R.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, N). On entry, the pentagonal M-by-N matrix B. The first M-L rows are rectangular, and the last L rows are upper trapezoidal. On exit, B contains the pentagonal matrix V. See Further Details.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M).

T Output parameter.

T is REAL

T is an array, dimension (LDT, N). The N-by-N upper triangular factor T of the block reflector. See Further Details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctpqrt2*, *dtpqrt2* and *ztpqrt2*. It also exists with a native C interface as *LAPACKE_stpqrt2*.

4.17.621 stprfb

stprfb applies a real "triangular-pentagonal" block reflector H or its conjugate transpose H^H to a real matrix C, which is composed of two blocks A and B, either from the left or right.

Syntax

Fortran specification:

```
use armpl_library

subroutine stprfb(SIDE, TRANS, DIRECT, STOREV, M, N, K, L, V, LDV, T, LDT, A,
                  LDA, B, LDB, WORK, LDWORK)
```

C specification:

```
#include "armpl.h"

void stprfb_(const char *side, const char *trans, const char *direct,
             const char *storev, const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *k, const armpl_int_t *l, const float *v,
             const armpl_int_t *ldv, const float *t, const armpl_int_t *ldt,
             float *a, const armpl_int_t *lda, float *b,
             const armpl_int_t *ldb, float *work, const armpl_int_t *ldwork,
             ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply H or H^H from the Left = 'R': apply H or H^H from the Right

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply H (No transpose) = 'C': apply H^H (Conjugate transpose)

DIRECT Input parameter.

DIRECT is CHARACTER*1

Indicates how H is formed from a product of elementary reflectors = 'F': H = H(1) H(2) . . . H(k) (Forward) = 'B': H = H(k) . . . H(2) H(1) (Backward)

STOREV Input parameter.

STOREV is CHARACTER*1

Indicates how the vectors which define the elementary reflectors are stored: = 'C': Columns = 'R': Rows

M Input parameter.

M is INTEGER

The number of rows of the matrix B. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix B. N >= 0.

K Input parameter.

K is INTEGER

The order of the matrix T, i.e. the number of elementary reflectors whose product defines the block reflector. K >= 0.

L Input parameter.

L is INTEGER

The order of the trapezoidal part of V. K >= L >= 0. See Further Details.

V Input parameter.

V is REAL

V is an array, dimension. (LDV, K) if STOREV = 'C' (LDV, M) if STOREV = 'R' and SIDE = 'L' (LDV, N) if STOREV = 'R' and SIDE = 'R' The pentagonal matrix V, which contains the elementary reflectors H(1), H(2), . . . , H(K). See Further Details.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. If STOREV = 'C' and SIDE = 'L', LDV >= max(1, M); if STOREV = 'C' and SIDE = 'R', LDV >= max(1, N); if STOREV = 'R', LDV >= K.

T Input parameter.

T is REAL

T is an array, dimension (LDT, K). The triangular K-by-K matrix T in the representation of the block reflector.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= K.

A Input and output parameter.

A is REAL

A is an array, dimension. (LDA, N) if SIDE = 'L' or (LDA, K) if SIDE = 'R' On entry, the K-by-N or M-by-K matrix A. On exit, A is overwritten by the corresponding block of H*C or H^H*C or C*H or C*H^H. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDC >= max(1, K); If SIDE = 'R', LDC >= max(1, M).

B Input and output parameter.

B is REAL

B is an array, dimension (LDB, N). On entry, the M-by-N matrix B. On exit, B is overwritten by the corresponding block of H*C or H^H*C or C*H or C*H^H. See Further Details.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M).

WORK Output parameter.

WORK is REAL

WORK is an array, dimension. (LDWORK, N) if SIDE = 'L', (LDWORK, K) if SIDE = 'R'.

LDWORK Input parameter.

LDWORK is INTEGER

The leading dimension of the array WORK. If SIDE = 'L', LDWORK >= K; if SIDE = 'R', LDWORK >= M.

Related Information

For this routine in other precisions, please see *ctprfb*, *dtprfb* and *ztprfb*. It also exists with a native C interface as *LAPACKE_stprfb*.

4.17.622 stpttf

stpttf copies a triangular matrix A from standard packed format (TP) to rectangular full packed format (TF).

Syntax

Fortran specification:

```
use armpl_library

subroutine stpttf(TRANSR, UPLO, N, AP, ARF, INFO)
```

C specification:

```
#include "armpl.h"

void stpttf_(const char *transr, const char *uplo, const armpl_int_t *n,
             const float *ap, float *arf, armpl_int_t *info, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': ARF in Normal format is wanted; = 'T': ARF in Conjugate-transpose format is wanted.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input parameter.

AP is REAL

AP is an array, dimension (N*(N+1)/2),. On entry, the upper or lower triangular matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

ARF Output parameter.

ARF is REAL

ARF is an array, dimension (N*(N+1)/2),. On exit, the upper or lower triangular matrix A stored in RFP format. For a further discussion see Notes below.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctpttf*, *dtpttf* and *ztpttf*. It also exists with a native C interface as *LAPACKE_stpttf*.

4.17.623 stpttr

stpttr copies a triangular matrix A from standard packed format (TP) to standard full format (TR).

Syntax

Fortran specification:

```
use armpl_library

subroutine stpttr(UPLO, N, AP, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void stpttr_(const char *uplo, const armpl_int_t *n, const float *ap,
             float *a, const armpl_int_t *lda, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular. = 'L': A is lower triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input parameter.

AP is REAL

AP is an array, dimension (N*(N+1)/2),. On entry, the upper or lower triangular matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

A Output parameter.

A is REAL

A is an array, dimension (LDA, N). On exit, the triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctpttr*, *dtpttr* and *ztpttr*. It also exists with a native C interface as *LAPACKE_stpttr*.

4.17.624 strti2

strti2 computes the inverse of a real upper or lower triangular matrix.

This is the Level 2 BLAS version of the algorithm.

Syntax

Fortran specification:

```
use armpl_library

subroutine strti2(UPLO, DIAG, N, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void strti2_(const char *uplo, const char *diag, const armpl_int_t *n,
             float *a, const armpl_int_t *lda, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the matrix A is upper or lower triangular. = 'U': Upper triangular = 'L': Lower triangular

DIAG Input parameter.

DIAG is CHARACTER*1

Specifies whether or not the matrix A is unit triangular. = 'N': Non-unit triangular = 'U': Unit triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is REAL

A is an array, dimension (LDA, N). On entry, the triangular matrix A. If UPLO = 'U', the leading n by n upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n by n lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1.

On exit, the (triangular) inverse of the original matrix, in the same storage format.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctrti2*, *dtrti2* and *ztrti2*.

4.17.625 strttf

strttf copies a triangular matrix A from standard full format (TR) to rectangular full packed format (TF) .

Syntax

Fortran specification:

```
use armpl_library

subroutine strttf(TRANSR, UPLO, N, A, LDA, ARF, INFO)
```

C specification:

```
#include "armpl.h"

void strttf_(const char *transr, const char *uplo, const armpl_int_t *n,
             const float *a, const armpl_int_t *lda, float *arf,
             armpl_int_t *info, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': ARF in Normal form is wanted; = 'T': ARF in Transpose form is wanted.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N).. On entry, the triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the matrix A. LDA >= max(1, N).

ARF Output parameter.

ARF is REAL

ARF is an array, dimension (NT).. NT=N*(N+1)/2. On exit, the triangular matrix A in RFP format.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctrttf*, *dtrttf* and *ztrttf*. It also exists with a native C interface as *LAPACKE_strttf*.

4.17.626 strttp

strttp copies a triangular matrix A from full format (TR) to standard packed format (TP).

Syntax

Fortran specification:

```
use armpl_library

subroutine strttp(UPLO, N, A, LDA, AP, INFO)
```

C specification:

```
#include "armpl.h"

void strttp_(const char *uplo, const armpl_int_t *n, const float *a,
             const armpl_int_t *lda, float *ap, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular. = 'L': A is lower triangular.

N Input parameter.

N is INTEGER

The order of the matrices AP and A. N >= 0.

A Input parameter.

A is REAL

A is an array, dimension (LDA, N). On exit, the triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AP Output parameter.

AP is REAL

AP is an array, dimension (N*(N+1)/2). On exit, the upper or lower triangular matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctrttp*, *dtrttp* and *ztrttp*. It also exists with a native C interface as *LAPACKE_strttp*.

4.17.627 zdrscl

zdrscl multiplies an n-element complex vector x by the real scalar 1/a. This is done without overflow or underflow as long as the final result x/a does not overflow or underflow.

Syntax

Fortran specification:

```
use armpl_library

subroutine zdrscl(N, SA, SX, INCX)
```

C specification:

```
#include "armpl.h"

void zdrscl_(const armpl_int_t *n, const double *sa,
             armpl_doublecomplex_t *sx, const armpl_int_t *incx);
```

Parameters

N Input parameter.

N is INTEGER

The number of components of the vector x.

SA Input parameter.

SA is DOUBLE PRECISION

The scalar a which is used to divide each component of x. SA must be >= 0, or the subroutine will divide by zero.

SX Input and output parameter.

SX is COMPLEX*16

SX is an array, dimension. (1+(N-1)*abs(INCX)) The n-element vector x.

INCX Input parameter.

INCX is INTEGER

The increment between successive values of the vector SX. > 0: SX(1) = X(1) and SX(1+(i-1)*INCX) = x(i), 1< i<= n

Related Information

4.17.628 zgbtf2

zgbtf2 computes an LU factorization of a complex m-by-n band matrix A using partial pivoting with row interchanges.

This is the unblocked version of the algorithm, calling Level 2 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgbtf2(M, N, KL, KU, AB, LDAB, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void zgbtf2_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *kl, const armpl_int_t *ku,
```

(continues on next page)

(continued from previous page)

```
            armpl_doublecomplex_t *ab, const armpl_int_t *ldab,
            armpl_int_t *ipiv, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

AB Input and output parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows KL+1 to 2*KL+KU+1; rows 1 to KL of the array need not be set. The j-th column of A is stored in the j-th column of the array AB as follows: AB(kl+ku+1+i-j,j) = A(i,j) for max(1,j-ku)<=i<=min(m,j+kl)

On exit, details of the factorization: U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1. See below for further details.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= 2*KL+KU+1.

IPIV Output parameter.

IPIV is INTEGER array, dimension (min(M, N))

The pivot indices; for 1 <= i <= min(M, N), row i of the matrix was interchanged with row IPIV(i).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value > 0: if INFO = +i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *cgbtf2*, *dgbtf2* and *sgbtf2*.

4.17.629 zgebd2

`zgebd2` reduces a complex general m by n matrix A to upper or lower real bidiagonal form B by a unitary transformation: Q^H * A * P = B.

If m >= n, B is upper bidiagonal; if m < n, B is lower bidiagonal.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgebd2(M, N, A, LDA, D, E, TAUQ, TAUP, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgebd2_(const armpl_int_t *m, const armpl_int_t *n,
             armpl_doublecomplex_t *a, const armpl_int_t *lda, double *d,
             double *e, armpl_doublecomplex_t *tauq,
             armpl_doublecomplex_t *taup, armpl_doublecomplex_t *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows in the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns in the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the m by n general matrix to be reduced. On exit, if m >= n, the diagonal and the first superdiagonal are overwritten with the upper bidiagonal matrix B; the elements below the diagonal, with the array TAUQ, represent the unitary matrix Q as a product of elementary reflectors, and the elements above the first superdiagonal, with the array TAUP, represent the unitary matrix P as a product of elementary reflectors; if m < n, the diagonal and the first subdiagonal are overwritten with the lower bidiagonal matrix B; the elements below the first subdiagonal, with the array TAUQ, represent the unitary matrix Q as a product of elementary reflectors, and the elements above the diagonal, with the array TAUP, represent the unitary matrix P as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

D Output parameter.

D is DOUBLE PRECISION

D is an array, dimension (min(M, N)). The diagonal elements of the bidiagonal matrix B: D(i) = A(i,i).

E Output parameter.

E is DOUBLE PRECISION

E is an array, dimension (min(M, N)-1). The off-diagonal elements of the bidiagonal matrix B: if m >= n, E(i) = A(i,i+1) for i = 1,2,. . . ,n-1; if m < n, E(i) = A(i+1,i) for i = 1,2,. . . ,m-1.

TAUQ Output parameter.

TAUQ is COMPLEX*16

TAUQ is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors which represent the unitary matrix Q. See Further Details.

TAUP Output parameter.

TAUP is COMPLEX*16

TAUP is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors which represent the unitary matrix P. See Further Details.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (max(M, N)) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cgebd2*, *dgebd2* and *sgebd2*.

4.17.630 zgehd2

zgehd2 reduces a complex general matrix A to upper Hessenberg form H by a unitary similarity transformation: $Q^H * A * Q = H$.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgehd2(N, ILO, IHI, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgehd2_(const armpl_int_t *n, const armpl_int_t *ilo,
             const armpl_int_t *ihi, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_doublecomplex_t *tau,
             armpl_doublecomplex_t *work, armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

It is assumed that A is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N. ILO and IHI are normally set by a previous call to ZGEBAL; otherwise they should be set to 1 and N respectively. See Further Details. 1 <= ILO <= IHI <= max(1, N).

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the n by n general matrix to be reduced. On exit, the upper triangle and the first subdiagonal of A are overwritten with the upper Hessenberg matrix H, and the elements below the first subdiagonal, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TAU Output parameter.

TAU is COMPLEX*16

TAU is an array, dimension (N-1). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cgehd2*, *dgehd2* and *sgehd2*.

4.17.631 zgelq2

zgelq2 computes an LQ factorization of a complex m by n matrix A: A = L * Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgelq2(M, N, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgelq2_(const armpl_int_t *m, const armpl_int_t *n,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_doublecomplex_t *tau, armpl_doublecomplex_t *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the m by n matrix A. On exit, the elements on and below the diagonal of the array contain the m by min(m,n) lower trapezoidal matrix L (L is lower triangular if m <= n); the elements above the diagonal, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is COMPLEX*16

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (M) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgelq2*, *dgelq2* and *sgelq2*. It also exists with a native C interface as *LAPACKE_zgelq2*.

4.17.632 zgelqt3

DGELQT3 recursively computes a LQ factorization of a complex M-by-N matrix A, using the compact WY representation of Q.

Based on the algorithm of Elmroth and Gustavson, IBM J. Res. Develop. Vol 44 No. 4 July 2000.

Syntax

Fortran specification:

```
use armpl_library

recursive subroutine zgelqt3(M, N, A, LDA, T, LDT, INFO)
```

C specification:

```
#include "armpl.h"

void zgelqt3_(const armpl_int_t *m, const armpl_int_t *n,
              armpl_doublecomplex_t *a, const armpl_int_t *lda,
              armpl_doublecomplex_t *t, const armpl_int_t *ldt,
              armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M =< N.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the real M-by-N matrix A. On exit, the elements on and below the diagonal contain the N-by-N lower triangular matrix L; the elements above the diagonal are the rows of V. See below for further details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

T Output parameter.

T is COMPLEX*16

T is an array, dimension (LDT, N). The N-by-N upper triangular factor of the block reflector. The elements on and above the diagonal contain the block reflector T; the elements below the diagonal are not used. See below for further details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgelqt3*, *dgelqt3* and *sgelqt3*.

4.17.633 zgeql2

zgeql2 computes a QL factorization of a complex m by n matrix A: A = Q * L.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgeql2(M, N, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgeql2_(const armpl_int_t *m, const armpl_int_t *n,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_doublecomplex_t *tau, armpl_doublecomplex_t *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the m by n matrix A. On exit, if m >= n, the lower triangle of the subarray A(m-n+1:m,1:n) contains the n by n lower triangular matrix L; if m <= n, the elements on and below the (n-m)-th superdiagonal contain the m by n lower trapezoidal matrix L; the remaining elements, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is COMPLEX*16

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgeql2*, *dgeql2* and *sgeql2*.

4.17.634 zgeqr2

zgeqr2 computes a QR factorization of a complex m by n matrix A: A = Q * R.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgeqr2(M, N, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgeqr2_(const armpl_int_t *m, const armpl_int_t *n,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_doublecomplex_t *tau, armpl_doublecomplex_t *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the m by n matrix A. On exit, the elements on and above the diagonal of the array contain the min(m,n) by n upper trapezoidal matrix R (R is upper triangular if m >=

n); the elements below the diagonal, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is COMPLEX*16

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgeqr2*, *dgeqr2* and *sgeqr2*. It also exists with a native C interface as *LAPACKE_zgeqr2*.

4.17.635 zgeqr2p

zgeqr2p computes a QR factorization of a complex m by n matrix A: A = Q * R. The diagonal entries of R are real and nonnegative.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgeqr2p(M, N, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgeqr2p_(const armpl_int_t *m, const armpl_int_t *n,
              armpl_doublecomplex_t *a, const armpl_int_t *lda,
              armpl_doublecomplex_t *tau, armpl_doublecomplex_t *work,
              armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the m by n matrix A. On exit, the elements on and above the diagonal of the array contain the min(m,n) by n upper trapezoidal matrix R (R is upper triangular if m >= n). The diagonal entries of R are real and nonnegative; the elements below the diagonal, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is COMPLEX*16

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgeqr2p*, *dgeqr2p* and *sgeqr2p*.

4.17.636 zgeqrt2

zgeqrt2 computes a QR factorization of a complex M-by-N matrix A, using the compact WY representation of Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgeqrt2(M, N, A, LDA, T, LDT, INFO)
```

C specification:

```
#include "armpl.h"

void zgeqrt2_(const armpl_int_t *m, const armpl_int_t *n,
              armpl_doublecomplex_t *a, const armpl_int_t *lda,
              armpl_doublecomplex_t *t, const armpl_int_t *ldt,
              armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= N.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the complex M-by-N matrix A. On exit, the elements on and above the diagonal contain the N-by-N upper triangular matrix R; the elements below the diagonal are the columns of V. See below for further details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

T Output parameter.

T is COMPLEX*16

T is an array, dimension (LDT, N). The N-by-N upper triangular factor of the block reflector. The elements on and above the diagonal contain the block reflector T; the elements below the diagonal are not used. See below for further details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgeqrt2*, *dgeqrt2* and *sgeqrt2*. It also exists with a native C interface as *LAPACKE_zgeqrt2*.

4.17.637 zgeqrt3

zgeqrt3 recursively computes a QR factorization of a complex M-by-N matrix A, using the compact WY representation of Q.

Based on the algorithm of Elmroth and Gustavson, IBM J. Res. Develop. Vol 44 No. 4 July 2000.

Syntax

Fortran specification:

```
use armpl_library

recursive subroutine zgeqrt3(M, N, A, LDA, T, LDT, INFO)
```

C specification:

```
#include "armpl.h"

void zgeqrt3_(const armpl_int_t *m, const armpl_int_t *n,
              armpl_doublecomplex_t *a, const armpl_int_t *lda,
              armpl_doublecomplex_t *t, const armpl_int_t *ldt,
              armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= N.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the complex M-by-N matrix A. On exit, the elements on and above the diagonal contain the N-by-N upper triangular matrix R; the elements below the diagonal are the columns of V. See below for further details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

T Output parameter.

T is COMPLEX*16

T is an array, dimension (LDT, N). The N-by-N upper triangular factor of the block reflector. The elements on and above the diagonal contain the block reflector T; the elements below the diagonal are not used. See below for further details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgeqrt3*, *dgeqrt3* and *sgeqrt3*. It also exists with a native C interface as *LAPACKE_zgeqrt3*.

4.17.638 zgerq2

zgerq2 computes an RQ factorization of a complex m by n matrix A: A = R * Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgerq2(M, N, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgerq2_(const armpl_int_t *m, const armpl_int_t *n,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_doublecomplex_t *tau, armpl_doublecomplex_t *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the m by n matrix A. On exit, if m <= n, the upper triangle of the subarray A(1:m,n-m+1:n) contains the m by m upper triangular matrix R; if m >= n, the elements on and above the (m-n)-th subdiagonal contain the m by n upper trapezoidal matrix R; the remaining elements, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is COMPLEX*16

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors (see Further Details).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (M) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgerq2*, *dgerq2* and *sgerq2*.

4.17.639 zgesc2

zgesc2 solves a system of linear equations

```
A * X = scale* RHS
```

with a general N-by-N matrix A using the LU factorization with complete pivoting computed by ZGETC2.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgesc2(N, A, LDA, RHS, IPIV, JPIV, SCALE)
```

C specification:

```
#include "armpl.h"

void zgesc2_(const armpl_int_t *n, const armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_doublecomplex_t *rhs,
             const armpl_int_t *ipiv, const armpl_int_t *jpiv,
             double *scale);
```

Parameters

N Input parameter.

N is INTEGER

The number of columns of the matrix A.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the LU part of the factorization of the n-by-n matrix A computed by ZGETC2: A = P * L * U * Q

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

RHS Input and output parameter.

RHS is COMPLEX*16

RHS is an array, dimension N.. On entry, the right hand side vector b. On exit, the solution vector X.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N).

The pivot indices; for 1 <= i <= N, row i of the matrix has been interchanged with row IPIV(i).

JPIV Input parameter.

JPIV is INTEGER array, dimension (N).

The pivot indices; for 1 <= j <= N, column j of the matrix has been interchanged with column JPIV(j).

SCALE Output parameter.

SCALE is DOUBLE PRECISION

On exit, SCALE contains the scale factor. SCALE is chosen 0 <= SCALE <= 1 to prevent owerflow in the solution.

Related Information

For this routine in other precisions, please see *cgesc2*, *dgesc2* and *sgesc2*.

4.17.640 zgetc2

zgetc2 computes an LU factorization, using complete pivoting, of the n-by-n matrix A. The factorization has the form A = P * L * U * Q, where P and Q are permutation matrices, L is lower triangular with unit diagonal elements and U is upper triangular.

This is a level 1 BLAS version of the algorithm.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgetc2(N, A, LDA, IPIV, JPIV, INFO)
```

C specification:

```
#include "armpl.h"

void zgetc2_(const armpl_int_t *n, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *ipiv, armpl_int_t *jpiv,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the n-by-n matrix to be factored. On exit, the factors L and U from the factorization A = P*L*U*Q; the unit diagonal elements of L are not stored. If U(k, k) appears to be less than SMIN, U(k, k) is given the value of SMIN, giving a nonsingular perturbed system.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N).

The pivot indices; for 1 <= i <= N, row i of the matrix has been interchanged with row IPIV(i).

JPIV Output parameter.

JPIV is INTEGER array, dimension (N).

The pivot indices; for 1 <= j <= N, column j of the matrix has been interchanged with column JPIV(j).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit > 0: if INFO = k, U(k, k) is likely to produce overflow if one tries to solve for x in Ax = b. So U is perturbed to avoid the overflow.

Related Information

For this routine in other precisions, please see *cgetc2*, *dgetc2* and *sgetc2*.

4.17.641 zgetf2

zgetf2 computes an LU factorization of a general m-by-n matrix A using partial pivoting with row interchanges.

The factorization has the form

```
A = P * L * U
```

where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if m > n), and U is upper triangular (upper trapezoidal if m < n).

This is the right-looking Level 2 BLAS version of the algorithm.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgetf2(M, N, A, LDA, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void zgetf2_(const armpl_int_t *m, const armpl_int_t *n,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_int_t *ipiv, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the m by n matrix to be factored. On exit, the factors L and U from the factorization A = P*L*U; the unit diagonal elements of L are not stored.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

IPIV Output parameter.

IPIV is INTEGER array, dimension (min(M, N))

The pivot indices; for 1 <= i <= min(M, N), row i of the matrix was interchanged with row IPIV(i).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value > 0: if INFO = k, U(k,k) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *cgetf2*, *dgetf2* and *sgetf2*. It also exists with a native C interface as *LAPACKE_zgetf2*.

4.17.642 zgsvj0

zgsvj0 is called from ZGESVJ as a pre-processor and that is its main purpose. It applies Jacobi rotations in the same way as ZGESVJ does, but it does not check convergence (stopping criterion). Few tuning parameters (marked by [TP]) are available for the implementer.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgsvj0(JOBV, M, N, A, LDA, D, SVA, MV, V, LDV, EPS, SFMIN, TOL,
                  NSWEEP, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgsvj0_(const char *jobv, const armpl_int_t *m, const armpl_int_t *n,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_doublecomplex_t *d, double *sva, const armpl_int_t *mv,
             armpl_doublecomplex_t *v, const armpl_int_t *ldv,
             const double *eps, const double *sfmin, const double *tol,
             const armpl_int_t *nsweep, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info, ... );
```

Parameters

JOBV Input parameter.

JOBV is CHARACTER*1

Specifies whether the output from this procedure is used to compute the matrix V: = 'V': the product of the Jacobi rotations is accumulated by postmulyiplying the N-by-N array V. (See the description of V.) = 'A': the product of the Jacobi rotations is accumulated by postmulyiplying the MV-by-N array V. (See the descriptions of MV and V.) = 'N': the Jacobi rotations are not accumulated.

M Input parameter.

M is INTEGER

The number of rows of the input matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the input matrix A. M >= N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, M-by-N matrix A, such that A*diag(D) represents the input matrix. On exit, A_onexit * diag(D_onexit) represents the input matrix A*diag(D) post-multiplied by a sequence of Jacobi rotations, where the rotation threshold and the total number of sweeps are given in TOL and NSWEEP, respectively. (See the descriptions of D, TOL and NSWEEP.)

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

D Input and output parameter.

D is COMPLEX*16

D is an array, dimension (N). The array D accumulates the scaling factors from the complex scaled Jacobi rotations. On entry, A*diag(D) represents the input matrix. On exit, A_onexit*diag(D_onexit) represents the input matrix post-multiplied by a sequence of Jacobi rotations, where the rotation threshold and the total number of sweeps are given in TOL and NSWEEP, respectively. (See the descriptions of A, TOL and NSWEEP.)

SVA Input and output parameter.

SVA is DOUBLE PRECISION

SVA is an array, dimension (N). On entry, SVA contains the Euclidean norms of the columns of the matrix A*diag(D). On exit, SVA contains the Euclidean norms of the columns of the matrix A_onexit*diag(D_onexit).

MV Input parameter.

MV is INTEGER

If JOBV .EQ. 'A', then MV rows of V are post-multipled by a sequence of Jacobi rotations. If JOBV = 'N', then MV is not referenced.

V Input and output parameter.

V is COMPLEX*16

V is an array, dimension (LDV, N). If JOBV .EQ. 'V' then N rows of V are post-multipled by a sequence of Jacobi rotations. If JOBV .EQ. 'A' then MV rows of V are post-multipled by a sequence of Jacobi rotations. If JOBV = 'N', then V is not referenced.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V, LDV >= 1. If JOBV = 'V', LDV .GE. N. If JOBV = 'A', LDV .GE. MV.

EPS Input parameter.

EPS is DOUBLE PRECISION

EPS = DLAMCH('Epsilon')

SFMIN Input parameter.

SFMIN is DOUBLE PRECISION

SFMIN = DLAMCH('Safe Minimum')

TOL Input parameter.

TOL is DOUBLE PRECISION

TOL is the threshold for Jacobi rotations. For a pair A(:,p), A(:,q) of pivot columns, the Jacobi rotation is applied only if ABS(COS(angle(A(:,p),A(:,q)))) .GT. TOL.

NSWEEP Input parameter.

NSWEEP is INTEGER

NSWEEP is the number of sweeps of Jacobi rotations to be performed.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

LWORK is the dimension of WORK. LWORK .GE. M.

INFO Output parameter.

INFO is INTEGER

= 0 : successful exit. < 0 : if INFO = -i, then the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgsvj0*, *dgsvj0* and *sgsvj0*.

4.17.643 zgsvj1

zgsvj1 is called from ZGESVJ as a pre-processor and that is its main purpose. It applies Jacobi rotations in the same way as ZGESVJ does, but it targets only particular pivots and it does not check convergence (stopping criterion). Few tunning parameters (marked by [TP]) are available for the implementer.

Further Details

zgsvj1 applies few sweeps of Jacobi rotations in the column space of the input M-by-N matrix A. The pivot pairs are taken from the (1,2) off-diagonal block in the corresponding N-by-N Gram matrix A^T * A. The block-entries (tiles) of the (1,2) off-diagonal block are marked by the [x]'s in the following scheme:

```
| *  *  * [x] [x] [x]|
| *  *  * [x] [x] [x]|    Row-cycling in the nblr-by-nblc [x] blocks.
| *  *  * [x] [x] [x]|    Row-cyclic pivoting inside each [x] block.
|[x] [x] [x] *  *  * |
|[x] [x] [x] *  *  * |
|[x] [x] [x] *  *  * |
```

In terms of the columns of A, the first N1 columns are rotated 'against' the remaining N-N1 columns, trying to increase the angle between the corresponding subspaces. The off-diagonal block is N1-by(N-N1) and it is tiled using quadratic tiles of side KBL. Here, KBL is a tunning parmeter. The number of sweeps is given in NSWEEP and the orthogonality threshold is given in TOL.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgsvj1(JOBV, M, N, N1, A, LDA, D, SVA, MV, V, LDV, EPS, SFMIN, TOL,
                  NSWEEP, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zgsvj1_(const char *jobv, const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *n1, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_doublecomplex_t *d, double *sva,
             const armpl_int_t *mv, armpl_doublecomplex_t *v,
             const armpl_int_t *ldv, const double *eps, const double *sfmin,
             const double *tol, const armpl_int_t *nsweep,
             armpl_doublecomplex_t *work, const armpl_int_t *lwork,
             armpl_int_t *info, ... );
```

Parameters

JOBV Input parameter.

JOBV is CHARACTER*1

Specifies whether the output from this procedure is used to compute the matrix V: = 'V': the product of the Jacobi rotations is accumulated by postmulyiplying the N-by-N array V. (See the description of V.) = 'A': the product of the Jacobi rotations is accumulated by postmulyiplying the MV-by-N array V. (See the descriptions of MV and V.) = 'N': the Jacobi rotations are not accumulated.

M Input parameter.

M is INTEGER

The number of rows of the input matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the input matrix A. M >= N >= 0.

N1 Input parameter.

N1 is INTEGER

N1 specifies the 2 x 2 block partition, the first N1 columns are rotated 'against' the remaining N-N1 columns of A.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, M-by-N matrix A, such that A*diag(D) represents the input matrix. On exit, A_onexit * D_onexit represents the input matrix A*diag(D) post-multiplied by a sequence of Jacobi rotations, where the rotation threshold and the total number of sweeps are given in TOL and NSWEEP, respectively. (See the descriptions of N1, D, TOL and NSWEEP.)

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

D Input and output parameter.

D is COMPLEX*16

D is an array, dimension (N). The array D accumulates the scaling factors from the fast scaled Jacobi rotations. On entry, A*diag(D) represents the input matrix. On exit, A_onexit*diag(D_onexit) represents the input matrix post-multiplied by a sequence of Jacobi rotations, where the rotation threshold and the total number of sweeps are given in TOL and NSWEEP, respectively. (See the descriptions of N1, A, TOL and NSWEEP.)

SVA Input and output parameter.

SVA is DOUBLE PRECISION

SVA is an array, dimension (N). On entry, SVA contains the Euclidean norms of the columns of the matrix A*diag(D). On exit, SVA contains the Euclidean norms of the columns of the matrix onexit*diag(D_onexit).

MV Input parameter.

MV is INTEGER

If JOBV .EQ. 'A', then MV rows of V are post-multipled by a sequence of Jacobi rotations. If JOBV = 'N', then MV is not referenced.

V Input and output parameter.

V is COMPLEX*16

V is an array, dimension (LDV, N). If JOBV .EQ. 'V' then N rows of V are post-multipled by a sequence of Jacobi rotations. If JOBV .EQ. 'A' then MV rows of V are post-multipled by a sequence of Jacobi rotations. If JOBV = 'N', then V is not referenced.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V, LDV >= 1. If JOBV = 'V', LDV .GE. N. If JOBV = 'A', LDV .GE. MV.

EPS Input parameter.

EPS is DOUBLE PRECISION

EPS = DLAMCH('Epsilon')

SFMIN Input parameter.

SFMIN is DOUBLE PRECISION

SFMIN = DLAMCH('Safe Minimum')

TOL Input parameter.

TOL is DOUBLE PRECISION

TOL is the threshold for Jacobi rotations. For a pair A(:,p), A(:,q) of pivot columns, the Jacobi rotation is applied only if ABS(COS(angle(A(:,p),A(:,q)))) .GT. TOL.

NSWEEP Input parameter.

NSWEEP is INTEGER

NSWEEP is the number of sweeps of Jacobi rotations to be performed.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

LWORK is the dimension of WORK. LWORK .GE. M.

INFO Output parameter.

INFO is INTEGER

= 0 : successful exit. < 0 : if INFO = -i, then the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cgsvj1*, *dgsvj1* and *sgsvj1*.

4.17.644 zgtts2

zgtts2 solves one of the systems of equations

```
A * X = B,  A**T * X = B,  or  A**H * X = B,
```

with a tridiagonal matrix A using the LU factorization computed by ZGTTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine zgtts2(ITRANS, N, NRHS, DL, D, DU, DU2, IPIV, B, LDB)
```

C specification:

```
#include "armpl.h"

void zgtts2_(const armpl_int_t *itrans, const armpl_int_t *n,
             const armpl_int_t *nrhs, const armpl_doublecomplex_t *dl,
             const armpl_doublecomplex_t *d, const armpl_doublecomplex_t *du,
             const armpl_doublecomplex_t *du2, const armpl_int_t *ipiv,
             armpl_doublecomplex_t *b, const armpl_int_t *ldb);
```

Parameters

ITRANS Input parameter.

ITRANS is INTEGER

Specifies the form of the system of equations. = 0: A * X = B (No transpose) = 1: A^T * X = B (Transpose) = 2: A^H * X = B (Conjugate transpose)

N Input parameter.

N is INTEGER

The order of the matrix A.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

DL Input parameter.

DL is COMPLEX*16

DL is an array, dimension (N-1). The (n-1) multipliers that define the matrix L from the LU factorization of A.

D Input parameter.

D is COMPLEX*16

D is an array, dimension (N). The n diagonal elements of the upper triangular matrix U from the LU factorization of A.

DU Input parameter.

DU is COMPLEX*16

DU is an array, dimension (N-1). The (n-1) elements of the first super-diagonal of U.

DU2 Input parameter.

DU2 is COMPLEX*16

DU2 is an array, dimension (N-2). The (n-2) elements of the second super-diagonal of U.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices; for 1 <= i <= n, row i of the matrix was interchanged with row IPIV(i). IPIV(i) will always be either i or i+1; IPIV(i) = i indicates a row interchange was not required.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the matrix of right hand side vectors B. On exit, B is overwritten by the solution vectors X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Related Information

For this routine in other precisions, please see *cgtts2*, *dgtts2* and *sgtts2*.

4.17.645 zhegs2

zhegs2 reduces a complex Hermitian-definite generalized eigenproblem to standard form.

If ITYPE = 1, the problem is A*x = lambda*B*x, and A is overwritten by inv(U^H)*A*inv(U) or inv(L)*A*inv(L^H)

If ITYPE = 2 or 3, the problem is A*B*x = lambda*x or B*A*x = lambda*x, and A is overwritten by U*A*U^H or L^H *A*L.

B must have been previously factorized as U^H *U or L*L^H by ZPOTRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhegs2(ITYPE, UPLO, N, A, LDA, B, LDB, INFO)
```

C specification:

```
#include "armpl.h"

void zhegs2_(const armpl_int_t *itype, const char *uplo, const armpl_int_t *n,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_doublecomplex_t *b, const armpl_int_t *ldb,
             armpl_int_t *info, ... );
```

Parameters

ITYPE Input parameter.

ITYPE is INTEGER

= 1: compute inv(U^H)*A*inv(U) or inv(L)*A*inv(L^H); = 2 or 3: compute U*A*U^H or L^H *A*L.

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the Hermitian matrix A is stored, and how B has been factorized. = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading n by n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n by n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the transformed matrix, stored in the same format as A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). The triangular factor from the Cholesky factorization of B, as returned by ZPOTRF.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *chegs2*.

4.17.646 zheswapr

ZHESWAPR applies an elementary permutation on the rows and the columns of a hermitian matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine zheswapr(UPLO, N, A, LDA, I1, I2)
```

C specification:

```
#include "armpl.h"

void zheswapr_(const char *uplo, const armpl_int_t *n,
               armpl_doublecomplex_t *a, const armpl_int_t *lda,
               const armpl_int_t *i1, const armpl_int_t *i2, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the NB diagonal matrix D and the multipliers used to obtain the factor U or L as computed by CSYTRF.

On exit, if INFO = 0, the (symmetric) inverse of the original matrix. If UPLO = 'U', the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; if UPLO = 'L' the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

I1 Input parameter.

I1 is INTEGER

Index of the first row to swap

I2 Input parameter.

I2 is INTEGER

Index of the second row to swap

Related Information

For this routine in other precisions, please see *cheswapr*. It also exists with a native C interface as *LAPACKE_zheswapr*.

4.17.647 zhetd2

zhetd2 reduces a complex Hermitian matrix A to real symmetric tridiagonal form T by a unitary similarity transformation: $Q^H * A * Q = T$.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhetd2(UPLO, N, A, LDA, D, E, TAU, INFO)
```

C specification:

```
#include "armpl.h"

void zhetd2_(const char *uplo, const armpl_int_t *n, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, double *d, double *e,
             armpl_doublecomplex_t *tau, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the Hermitian matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, if UPLO = 'U', the diagonal and first superdiagonal of A are overwritten by the corresponding elements of the tridiagonal matrix T, and the elements above the first superdiagonal, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors; if UPLO = 'L', the diagonal and first subdiagonal of A are over- written by the corresponding elements of the tridiagonal matrix T, and the elements below the first subdiagonal, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

D Output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The diagonal elements of the tridiagonal matrix T: D(i) = A(i,i).

E Output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N-1). The off-diagonal elements of the tridiagonal matrix T: E(i) = A(i,i+1) if UPLO = 'U', E(i) = A(i+1,i) if UPLO = 'L'.

TAU Output parameter.

TAU is COMPLEX*16

TAU is an array, dimension (N-1). The scalar factors of the elementary reflectors (see Further Details).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *chetd2*.

4.17.648 zhetf2

zhetf2 computes the factorization of a complex Hermitian matrix A using the Bunch-Kaufman diagonal pivoting method:

```
A = U*D*U**H  or  A = L*D*L**H
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, U^H is the conjugate transpose of U, and D is Hermitian and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the unblocked version of the algorithm, calling Level 2 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhetf2(UPLO, N, A, LDA, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void zhetf2_(const char *uplo, const armpl_int_t *n, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *ipiv, armpl_int_t *info,
             ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the Hermitian matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L (see below for further details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D.

If UPLO = 'U': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block.

If UPLO = 'L': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value > 0: if INFO = k, D(k,k) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *chetf2*.

4.17.649 zhetf2_rook

ZHETF2_ROOK computes the factorization of a complex Hermitian matrix A using the bounded Bunch-Kaufman ("rook") diagonal pivoting method:

```
A = U*D*U**H  or  A = L*D*L**H
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, U^H is the conjugate transpose of U, and D is Hermitian and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the unblocked version of the algorithm, calling Level 2 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhetf2_rook(UPLO, N, A, LDA, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void zhetf2_rook_(const char *uplo, const armpl_int_t *n,
                  armpl_doublecomplex_t *a, const armpl_int_t *lda,
                  armpl_int_t *ipiv, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the Hermitian matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L (see below for further details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D.

If UPLO = 'U': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k-1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k-1 and -IPIV(k-1) were inerchaged, D(k-1:k,k-1:k) is a 2-by-2 diagonal block.

If UPLO = 'L': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k+1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k+1 and -IPIV(k+1) were inerchaged, D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value > 0: if INFO = k, D(k,k) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *chetf2_rook*.

4.17.650 zhfrk

Level 3 BLAS like routine for C in RFP Format.

zhfrk performs one of the Hermitian rank–k operations

```
C := alpha*A*A**H + beta*C,
```

or

```
C := alpha*A**H*A + beta*C,
```

where alpha and beta are real scalars, C is an n–by–n Hermitian matrix and A is an n–by–k matrix in the first case and a k–by–n matrix in the second case.

Syntax

Fortran specification:

```
use armpl_library

subroutine zhfrk(TRANSR, UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C)
```

C specification:

```
#include "armpl.h"

void zhfrk_(const char *transr, const char *uplo, const char *trans,
            const armpl_int_t *n, const armpl_int_t *k, const double *alpha,
            const armpl_doublecomplex_t *a, const armpl_int_t *lda,
            const double *beta, armpl_doublecomplex_t *c, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': The Normal Form of RFP A is stored; = 'C': The Conjugate-transpose Form of RFP A is stored.

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the upper or lower triangular part of the array C is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of C is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of C is to be referenced.

Unchanged on exit.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' C := alpha*A*A^H + beta*C.

TRANS = 'C' or 'c' C := alpha*A^H * A + beta*C.

Unchanged on exit.

N Input parameter.

N is INTEGER

On entry, N specifies the order of the matrix C. N must be at least zero. Unchanged on exit.

K Input parameter.

K is INTEGER

On entry with TRANS = 'N' or 'n', K specifies the number of columns of the matrix A, and on entry with TRANS = 'C' or 'c', K specifies the number of rows of the matrix A. K must be at least zero. Unchanged on exit.

ALPHA Input parameter.

ALPHA is DOUBLE PRECISION

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA,ka). where KA is K when TRANS = 'N' or 'n', and is N otherwise. Before entry with TRANS = 'N' or 'n', the leading N–by–K part of the array A must contain the matrix A, otherwise the leading K–by–N part of the array A must contain the matrix A. Unchanged on exit.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDA must be at least max(1, n), otherwise LDA must be at least max(1, k). Unchanged on exit.

BETA Input parameter.

BETA is DOUBLE PRECISION

On entry, BETA specifies the scalar beta. Unchanged on exit.

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (N*(N+1)/2). On entry, the matrix A in RFP Format. RFP Format is described by TRANSR, UPLO and N. Note that the imaginary parts of the diagonal elements need not be set, they are assumed to be zero, and on exit they are set to zero.

Related Information

For this routine in other precisions, please see *chfrk*. It also exists with a native C interface as *LAPACKE_zhfrk*.

4.17.651 zla_gbamv

ZLA_GBAMV performs one of the matrix-vector operations

```
      y := alpha*abs(A)*abs(x) + beta*abs(y),
or    y := alpha*abs(A)**T*abs(x) + beta*abs(y),
```

where alpha and beta are scalars, x and y are vectors and A is an m by n matrix.

This function is primarily used in calculating error bounds. To protect against underflow during evaluation, components in the resulting vector are perturbed away from zero by (N+1) times the underflow threshold. To prevent unnecessarily large errors for block-structure embedded in general matrices, "symbolically" zero components are not perturbed. A zero entry is considered "symbolic" if all multiplications involved in computing that entry have at least one zero multiplicand.

Syntax

Fortran specification:

```
use armpl_library

subroutine zla_gbamv(TRANS, M, N, KL, KU, ALPHA, AB, LDAB, X, INCX, BETA, Y,
                     INCY)
```

C specification:

```
#include "armpl.h"

void zla_gbamv_(const armpl_int_t *trans, const armpl_int_t *m,
                const armpl_int_t *n, const armpl_int_t *kl,
                const armpl_int_t *ku, const double *alpha,
                const armpl_doublecomplex_t *ab, const armpl_int_t *ldab,
                const armpl_doublecomplex_t *x, const armpl_int_t *incx,
                const double *beta, double *y, const armpl_int_t *incy);
```

Parameters

TRANS Input parameter.

TRANS is INTEGER

On entry, TRANS specifies the operation to be performed as follows:

BLAS_NO_TRANS y := alpha*abs(A)*abs(x) + beta*abs(y) BLAS_TRANS y := alpha*abs(A^T)*abs(x) + beta*abs(y) BLAS_CONJ_TRANS y := alpha*abs(A^T)*abs(x) + beta*abs(y)

Unchanged on exit.

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of the matrix A. M must be at least zero. Unchanged on exit.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix A. N must be at least zero. Unchanged on exit.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

ALPHA Input parameter.

ALPHA is DOUBLE PRECISION

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

AB Input parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, n). Before entry, the leading m by n part of the array AB must contain the matrix of coefficients. Unchanged on exit.

LDAB Input parameter.

LDAB is INTEGER

On entry, LDAB specifies the first dimension of AB as declared in the calling (sub) program. LDAB must be at least max(1, m). Unchanged on exit.

X Input parameter.

X is COMPLEX*16

X is an array, dimension. (1 + (n - 1)*abs(INCX)) when TRANS = 'N' or 'n' and at least (1 + (m - 1)*abs(INCX)) otherwise. Before entry, the incremented array X must contain the vector x. Unchanged on exit.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

BETA Input parameter.

BETA is DOUBLE PRECISION

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input. Unchanged on exit.

Y Input and output parameter.

Y is DOUBLE PRECISION

Y is an array, dimension. (1 + (m - 1)*abs(INCY)) when TRANS = 'N' or 'n' and at least (1 + (n - 1)*abs(INCY)) otherwise. Before entry with BETA non-zero, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero. Unchanged on exit.

Level 2 Blas routine.

Related Information

For this routine in other precisions, please see *cla_gbamv*, *dla_gbamv* and *sla_gbamv*.

4.17.652 zla_gbrcond_c

```
ZLA_GBRCOND_C Computes the infinity norm condition number of
   op(A) * inv(diag(C)) where C is a DOUBLE PRECISION vector.
```

Syntax

Fortran specification:

```
use armpl_library

double precision function zla_gbrcond_c(TRANS, N, KL, KU, AB, LDAB, AFB,
                                        LDAFB, IPIV, C, CAPPLY, INFO, WORK,
                                        RWORK)
```

C specification:

```
#include "armpl.h"

double zla_gbrcond_c_(const char *trans, const armpl_int_t *n,
                      const armpl_int_t *kl, const armpl_int_t *ku,
                      const armpl_doublecomplex_t *ab,
                      const armpl_int_t *ldab,
                      const armpl_doublecomplex_t *afb,
                      const armpl_int_t *ldafb, const armpl_int_t *ipiv,
                      const double *c, const armpl_int_t *capply,
                      armpl_int_t *info, const armpl_doublecomplex_t *work,
                      const double *rwork, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate Transpose = Transpose)

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

AB Input parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(KU+1+i-j,j) = A(i,j) for max(1,j-KU)<=i<=min(N,j+kl)

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KL+KU+1.

AFB Input parameter.

AFB is COMPLEX*16

AFB is an array, dimension (LDAFB, N). Details of the LU factorization of the band matrix A, as computed by ZGBTRF. U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1.

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AFB. LDAFB >= 2*KL+KU+1.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from the factorization A = P*L*U as computed by ZGBTRF; row i of the matrix was interchanged with row IPIV(i).

C Input parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). The vector C in the formula op(A) * inv(diag(C)).

CAPPLY Input parameter.

CAPPLY is LOGICAL

If .TRUE. then access the vector C in the formula above.

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. i > 0: The ith argument is invalid.

WORK Input parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N).. Workspace.

RWORK Input parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N).. Workspace.

Related Information

For this routine in other precisions, please see *cla_gbrcond_c*.

4.17.653 zla_gbrcond_x

```
ZLA_GBRCOND_X Computes the infinity norm condition number of
   op(A) * diag(X) where X is a COMPLEX*16 vector.
```

Syntax

Fortran specification:

```
use armpl_library

double precision function zla_gbrcond_x(TRANS, N, KL, KU, AB, LDAB, AFB,
                                        LDAFB, IPIV, X, INFO, WORK, RWORK)
```

C specification:

```
#include "armpl.h"

double zla_gbrcond_x_(const char *trans, const armpl_int_t *n,
                      const armpl_int_t *kl, const armpl_int_t *ku,
                      const armpl_doublecomplex_t *ab,
                      const armpl_int_t *ldab,
                      const armpl_doublecomplex_t *afb,
                      const armpl_int_t *ldafb, const armpl_int_t *ipiv,
                      const armpl_doublecomplex_t *x, armpl_int_t *info,
                      const armpl_doublecomplex_t *work, const double *rwork,
                      ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate Transpose = Transpose)

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

AB Input parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(KU+1+i-j,j) = A(i,j) for max(1,j-KU)<=i<=min(N,j+kl)

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KL+KU+1.

AFB Input parameter.

AFB is COMPLEX*16

AFB is an array, dimension (LDAFB, N). Details of the LU factorization of the band matrix A, as computed by ZGBTRF. U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1.

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AFB. LDAFB >= 2*KL+KU+1.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from the factorization A = P*L*U as computed by ZGBTRF; row i of the matrix was interchanged with row IPIV(i).

X Input parameter.

X is COMPLEX*16

X is an array, dimension (N). The vector X in the formula op(A) * diag(X).

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. i > 0: The ith argument is invalid.

WORK Input parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N).. Workspace.

RWORK Input parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N).. Workspace.

Related Information

For this routine in other precisions, please see *cla_gbrcond_x*.

4.17.654 zla_gbrfsx_extended

zla_gbrfsx_extended improves the computed solution to a system of linear equations by performing extra-precise iterative refinement and provides error bounds and backward error estimates for the solution. This subroutine is called by ZGBRFSX to perform iterative refinement. In addition to normwise error bound, the code provides maximum componentwise error bound if possible. See comments for ERR_BNDS_NORM and ERR_BNDS_COMP for details of the error bounds. Note that this subroutine is only resonsible for setting the second fields of ERR_BNDS_NORM and ERR_BNDS_COMP.

Syntax

Fortran specification:

```
use armpl_library

subroutine zla_gbrfsx_extended(PREC_TYPE, TRANS_TYPE, N, KL, KU, NRHS, AB,
                               LDAB, AFB, LDAFB, IPIV, COLEQU, C, B, LDB, Y,
                               LDY, BERR_OUT, N_NORMS, ERR_BNDS_NORM,
                               ERR_BNDS_COMP, RES, AYB, DY, Y_TAIL, RCOND,
                               ITHRESH, RTHRESH, DZ_UB, IGNORE_CWISE, INFO)
```

C specification:

```
#include "armpl.h"

void zla_gbrfsx_extended_(const armpl_int_t *prec_type,
                          const armpl_int_t *trans_type, const armpl_int_t *n,
                          const armpl_int_t *kl, const armpl_int_t *ku,
                          const armpl_int_t *nrhs,
                          const armpl_doublecomplex_t *ab,
                          const armpl_int_t *ldab,
                          const armpl_doublecomplex_t *afb,
                          const armpl_int_t *ldafb, const armpl_int_t *ipiv,
                          const armpl_int_t *colequ, const double *c,
                          const armpl_doublecomplex_t *b,
                          const armpl_int_t *ldb, armpl_doublecomplex_t *y,
                          const armpl_int_t *ldy, double *berr_out,
                          const armpl_int_t *n_norms, double *err_bnds_norm,
                          double *err_bnds_comp, armpl_doublecomplex_t *res,
                          double *ayb, armpl_doublecomplex_t *dy,
                          armpl_doublecomplex_t *y_tail, const double *rcond,
                          const armpl_int_t *ithresh, const double *rthresh,
                          const double *dz_ub,
                          const armpl_int_t *ignore_cwise,
                          armpl_int_t *info);
```

Parameters

PREC_TYPE Input parameter.

PREC_TYPE is INTEGER

Specifies the intermediate precision to be used in refinement. The value is defined by ILAPREC(P) where P is a CHARACTER and P = ‘S’: Single = ‘D’: Double = ‘I’: Indigenous = ‘X’, ‘E’: Extra

TRANS_TYPE Input parameter.

TRANS_TYPE is INTEGER

Specifies the transposition operation on A. The value is defined by ILATRANS(T) where T is a CHARACTER and T = ‘N’: No transpose = ‘T’: Transpose = ‘C’: Conjugate transpose

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0

NRHS Input parameter.

NRHS is INTEGER

The number of right-hand-sides, i.e., the number of columns of the matrix B.

AB Input parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). On entry, the N-by-N matrix A.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array A. LDAB >= max(1, N).

AFB Input parameter.

AFB is COMPLEX*16

AFB is an array, dimension (LDAF, N). The factors L and U from the factorization A = P*L*U as computed by ZGBTRF.

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from the factorization A = P*L*U as computed by ZGBTRF; row i of the matrix was interchanged with row IPIV(i).

COLEQU Input parameter.

COLEQU is LOGICAL

If .TRUE. then column equilibration was done to A before calling this routine. This is needed to compute the solution and error bounds correctly.

C Input parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). The column scale factors for A. If COLEQU = .FALSE., C is not accessed. If C is input, each element of C should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). The right-hand-side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Y Input and output parameter.

Y is COMPLEX*16

Y is an array, dimension (LDY, NRHS). On entry, the solution matrix X, as computed by ZGBTRS. On exit, the improved solution matrix Y.

LDY Input parameter.

LDY is INTEGER

The leading dimension of the array Y. LDY >= max(1, N).

BERR_OUT Output parameter.

BERR_OUT is DOUBLE PRECISION

BERR_OUT is an array, dimension (NRHS). On exit, BERR_OUT(j) contains the componentwise relative backward error for right-hand-side j from the formula max(i) (abs(RES(i)) / (abs(op(A_s))*abs(Y) + abs(B_s))(i)) where abs(Z) is the componentwise absolute value of the matrix or vector Z. This is computed by ZLA_LIN_BERR.

N_NORMS Input parameter.

N_NORMS is INTEGER

Determines which error bounds to return (see ERR_BNDS_NORM and ERR_BNDS_COMP). If N_NORMS >= 1 return normwise error bounds. If N_NORMS >= 2 return componentwise error bounds.

ERR_BNDS_NORM Input and output parameter.

ERR_BNDS_NORM is DOUBLE PRECISION

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 “Trust/don’t trust” boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch(‘Epsilon’).

err = 2 “Guaranteed” error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch(‘Epsilon’). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch(‘Epsilon’) to determine if the error estimate is “guaranteed”. These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

This subroutine is only responsible for setting the second field above. See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Input and output parameter.

ERR_BNDS_COMP is DOUBLE PRECISION

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:,N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

This subroutine is only responsible for setting the second field above. See Lapack Working Note 165 for further details and extra cautions.

RES Input parameter.

RES is COMPLEX*16

RES is an array, dimension (N). Workspace to hold the intermediate residual.

AYB Input parameter.

AYB is DOUBLE PRECISION

AYB is an array, dimension (N). Workspace.

DY Input parameter.

DY is COMPLEX*16

DY is an array, dimension (N). Workspace to hold the intermediate solution.

Y_TAIL Input parameter.

Y_TAIL is COMPLEX*16

Y_TAIL is an array, dimension (N). Workspace to hold the trailing bits of the intermediate solution.

RCOND Input parameter.

RCOND is DOUBLE PRECISION

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

ITHRESH Input parameter.

ITHRESH is INTEGER

The maximum number of residual computations allowed for refinement. The default is 10. For 'aggressive' set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in ERR_BNDS_NORM and ERR_BNDS_COMP may no longer be trustworthy.

RTHRESH Input parameter.

RTHRESH is DOUBLE PRECISION

Determines when to stop refinement if the error estimate stops decreasing. Refinement will stop when the next solution no longer satisfies norm(dx_{i+1}) < RTHRESH * norm(dx_i) where norm(Z) is the infinity norm of Z. RTHRESH satisfies 0 < RTHRESH <= 1. The default value is 0.5. For 'aggressive' set to 0.9 to permit convergence on extremely ill-conditioned matrices. See LAWN 165 for more details.

DZ_UB Input parameter.

DZ_UB is DOUBLE PRECISION

Determines when to start considering componentwise convergence. Componentwise convergence is only considered after each component of the solution Y is stable, which we definte as the relative change in each component being less than DZ_UB. The default value is 0.25, requiring the first bit to be stable. See LAWN 165 for more details.

IGNORE_CWISE Input parameter.

IGNORE_CWISE is LOGICAL

If .TRUE. then ignore componentwise convergence. Default value is .FALSE..

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. < 0: if INFO = -i, the ith argument to ZGBTRS had an illegal value

Related Information

For this routine in other precisions, please see *cla_gbrfsx_extended*, *dla_gbrfsx_extended* and *sla_gbrfsx_extended*.

4.17.655 zla_gbrpvgrw

ZLA_GBRPVGRW computes the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If this is much less than 1, the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, estimated condition numbers, and error bounds could be unreliable.

Syntax

Fortran specification:

```
use armpl_library

double precision function zla_gbrpvgrw(N, KL, KU, NCOLS, AB, LDAB, AFB,
                                       LDAFB)
```

C specification:

```
#include "armpl.h"

double zla_gbrpvgrw_(const armpl_int_t *n, const armpl_int_t *kl,
                     const armpl_int_t *ku, const armpl_int_t *ncols,
                     const armpl_doublecomplex_t *ab, const armpl_int_t *ldab,
                     const armpl_doublecomplex_t *afb,
                     const armpl_int_t *ldafb);
```

Parameters

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

NCOLS Input parameter.

NCOLS is INTEGER

The number of columns of the matrix A. NCOLS >= 0.

AB Input parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(KU+1+i-j,j) = A(i,j) for max(1,j-KU)<=i<=min(N,j+kl)

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KL+KU+1.

AFB Input parameter.

AFB is COMPLEX*16

AFB is an array, dimension (LDAFB, N). Details of the LU factorization of the band matrix A, as computed by ZGBTRF. U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1.

LDAFB Input parameter.

LDAFB is INTEGER

The leading dimension of the array AFB. LDAFB >= 2*KL+KU+1.

Related Information

For this routine in other precisions, please see *cla_gbrpvgrw*, *dla_gbrpvgrw* and *sla_gbrpvgrw*.

4.17.656 zla_geamv

ZLA_GEAMV performs one of the matrix-vector operations

```
     y := alpha*abs(A)*abs(x) + beta*abs(y),
or   y := alpha*abs(A)**T*abs(x) + beta*abs(y),
```

where alpha and beta are scalars, x and y are vectors and A is an m by n matrix.

This function is primarily used in calculating error bounds. To protect against underflow during evaluation, components in the resulting vector are perturbed away from zero by (N+1) times the underflow threshold. To prevent unnecessarily large errors for block-structure embedded in general matrices, "symbolically" zero components are not perturbed. A zero entry is considered "symbolic" if all multiplications involved in computing that entry have at least one zero multiplicand.

Syntax

Fortran specification:

```
use armpl_library

subroutine zla_geamv(TRANS, M, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
```

C specification:

```
#include "armpl.h"

void zla_geamv_(const armpl_int_t *trans, const armpl_int_t *m,
                const armpl_int_t *n, const double *alpha,
                const armpl_doublecomplex_t *a, const armpl_int_t *lda,
                const armpl_doublecomplex_t *x, const armpl_int_t *incx,
                const double *beta, double *y, const armpl_int_t *incy);
```

Parameters

TRANS Input parameter.

TRANS is INTEGER

On entry, TRANS specifies the operation to be performed as follows:

BLAS_NO_TRANS y := alpha*abs(A)*abs(x) + beta*abs(y) BLAS_TRANS y := alpha*abs(A^T)*abs(x) + beta*abs(y) BLAS_CONJ_TRANS y := alpha*abs(A^T)*abs(x) + beta*abs(y)

Unchanged on exit.

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of the matrix A. M must be at least zero. Unchanged on exit.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix A. N must be at least zero. Unchanged on exit.

ALPHA Input parameter.

ALPHA is DOUBLE PRECISION

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, n). Before entry, the leading m by n part of the array A must contain the matrix of coefficients. Unchanged on exit.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, m). Unchanged on exit.

X Input parameter.

X is COMPLEX*16

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)) when TRANS = 'N' or 'n' and at least (1 + (m - 1)*abs(INCX)) otherwise. Before entry, the incremented array X must contain the vector x. Unchanged on exit.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

BETA Input parameter.

BETA is DOUBLE PRECISION

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input. Unchanged on exit.

Y Input and output parameter.

Y is DOUBLE PRECISION

Y is an array, dimension. (1 + (m - 1)*abs(INCY)) when TRANS = 'N' or 'n' and at least (1 + (n - 1)*abs(INCY)) otherwise. Before entry with BETA non-zero, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero. Unchanged on exit.

Level 2 Blas routine.

Related Information

For this routine in other precisions, please see *cla_geamv*, *dla_geamv* and *sla_geamv*.

4.17.657 zla_gercond_c

```
ZLA_GERCOND_C computes the infinity norm condition number of
   op(A) * inv(diag(C)) where C is a DOUBLE PRECISION vector.
```

Syntax

Fortran specification:

```
use armpl_library

double precision function zla_gercond_c(TRANS, N, A, LDA, AF, LDAF, IPIV, C,
                                        CAPPLY, INFO, WORK, RWORK)
```

C specification:

```
#include "armpl.h"

double zla_gercond_c_(const char *trans, const armpl_int_t *n,
                      const armpl_doublecomplex_t *a, const armpl_int_t *lda,
                      const armpl_doublecomplex_t *af,
                      const armpl_int_t *ldaf, const armpl_int_t *ipiv,
                      const double *c, const armpl_int_t *capply,
                      armpl_int_t *info, const armpl_doublecomplex_t *work,
                      const double *rwork, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate Transpose = Transpose)

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX*16

AF is an array, dimension (LDAF, N). The factors L and U from the factorization A = P*L*U as computed by ZGETRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from the factorization A = P*L*U as computed by ZGETRF; row i of the matrix was interchanged with row IPIV(i).

C Input parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). The vector C in the formula op(A) * inv(diag(C)).

CAPPLY Input parameter.

CAPPLY is LOGICAL

If .TRUE. then access the vector C in the formula above.

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. i > 0: The ith argument is invalid.

WORK Input parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N).. Workspace.

RWORK Input parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N).. Workspace.

Related Information

For this routine in other precisions, please see *cla_gercond_c*.

4.17.658 zla_gercond_x

```
ZLA_GERCOND_X computes the infinity norm condition number of
   op(A) * diag(X) where X is a COMPLEX*16 vector.
```

Syntax

Fortran specification:

```
use armpl_library

double precision function zla_gercond_x(TRANS, N, A, LDA, AF, LDAF, IPIV, X,
                                        INFO, WORK, RWORK)
```

C specification:

```
#include "armpl.h"

double zla_gercond_x_(const char *trans, const armpl_int_t *n,
                      const armpl_doublecomplex_t *a, const armpl_int_t *lda,
                      const armpl_doublecomplex_t *af,
                      const armpl_int_t *ldaf, const armpl_int_t *ipiv,
                      const armpl_doublecomplex_t *x, armpl_int_t *info,
                      const armpl_doublecomplex_t *work, const double *rwork,
                      ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the form of the system of equations: = 'N': A * X = B (No transpose) = 'T': A^T * X = B (Transpose) = 'C': A^H * X = B (Conjugate Transpose = Transpose)

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX*16

AF is an array, dimension (LDAF, N). The factors L and U from the factorization A = P*L*U as computed by ZGETRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from the factorization A = P*L*U as computed by ZGETRF; row i of the matrix was interchanged with row IPIV(i).

X Input parameter.

X is COMPLEX*16

X is an array, dimension (N). The vector X in the formula op(A) * diag(X).

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. i > 0: The ith argument is invalid.

WORK Input parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N).. Workspace.

RWORK Input parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N).. Workspace.

Related Information

For this routine in other precisions, please see *cla_gercond_x*.

4.17.659 zla_gerfsx_extended

zla_gerfsx_extended improves the computed solution to a system of linear equations by performing extra-precise iterative refinement and provides error bounds and backward error estimates for the solution. This subroutine is called by ZGERFSX to perform iterative refinement. In addition to normwise error bound, the code provides maximum componentwise error bound if possible. See comments for ERRS_N and ERRS_C for details of the error bounds. Note that this subroutine is only resonsible for setting the second fields of ERRS_N and ERRS_C.

Syntax

Fortran specification:

```
use armpl_library

subroutine zla_gerfsx_extended(PREC_TYPE, TRANS_TYPE, N, NRHS, A, LDA, AF,
                               LDAF, IPIV, COLEQU, C, B, LDB, Y, LDY,
                               BERR_OUT, N_NORMS, ERRS_N, ERRS_C, RES, AYB,
                               DY, Y_TAIL, RCOND, ITHRESH, RTHRESH, DZ_UB,
                               IGNORE_CWISE, INFO)
```

C specification:

```
#include "armpl.h"

void zla_gerfsx_extended_(const armpl_int_t *prec_type,
                          const armpl_int_t *trans_type, const armpl_int_t *n,
                          const armpl_int_t *nrhs,
                          const armpl_doublecomplex_t *a,
                          const armpl_int_t *lda,
                          const armpl_doublecomplex_t *af,
                          const armpl_int_t *ldaf, const armpl_int_t *ipiv,
                          const armpl_int_t *colequ, const double *c,
                          const armpl_doublecomplex_t *b,
                          const armpl_int_t *ldb, armpl_doublecomplex_t *y,
                          const armpl_int_t *ldy, double *berr_out,
                          const armpl_int_t *n_norms, double *errs_n,
                          double *errs_c, armpl_doublecomplex_t *res,
                          double *ayb, armpl_doublecomplex_t *dy,
                          armpl_doublecomplex_t *y_tail, const double *rcond,
                          const armpl_int_t *ithresh, const double *rthresh,
                          const double *dz_ub,
                          const armpl_int_t *ignore_cwise,
                          armpl_int_t *info);
```

Parameters

PREC_TYPE Input parameter.

PREC_TYPE is INTEGER

Specifies the intermediate precision to be used in refinement. The value is defined by ILAPREC(P) where P is a CHARACTER and P = 'S': Single = 'D': Double = 'I': Indigenous = 'X', 'E': Extra

TRANS_TYPE Input parameter.

TRANS_TYPE is INTEGER

Specifies the transposition operation on A. The value is defined by ILATRANS(T) where T is a CHARACTER and T = 'N': No transpose = 'T': Transpose = 'C': Conjugate transpose

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right-hand-sides, i.e., the number of columns of the matrix B.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX*16

AF is an array, dimension (LDAF, N). The factors L and U from the factorization A = P*L*U as computed by ZGETRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

The pivot indices from the factorization A = P*L*U as computed by ZGETRF; row i of the matrix was interchanged with row IPIV(i).

COLEQU Input parameter.

COLEQU is LOGICAL

If .TRUE. then column equilibration was done to A before calling this routine. This is needed to compute the solution and error bounds correctly.

C Input parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). The column scale factors for A. If COLEQU = .FALSE., C is not accessed. If C is input, each element of C should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). The right-hand-side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Y Input and output parameter.

Y is COMPLEX*16

Y is an array, dimension (LDY, NRHS). On entry, the solution matrix X, as computed by ZGETRS. On exit, the improved solution matrix Y.

LDY Input parameter.

LDY is INTEGER

The leading dimension of the array Y. LDY >= max(1, N).

BERR_OUT Output parameter.

BERR_OUT is DOUBLE PRECISION

BERR_OUT is an array, dimension (NRHS). On exit, BERR_OUT(j) contains the componentwise relative backward error for right-hand-side j from the formula max(i) (abs(RES(i)) / (abs(op(A_s))*abs(Y) + abs(B_s))(i)) where abs(Z) is the componentwise absolute value of the matrix or vector Z. This is computed by ZLA_LIN_BERR.

N_NORMS Input parameter.

N_NORMS is INTEGER

Determines which error bounds to return (see ERRS_N and ERRS_C). If N_NORMS >= 1 return normwise error bounds. If N_NORMS >= 2 return componentwise error bounds.

ERRS_N Input and output parameter.

ERRS_N is DOUBLE PRECISION

ERRS_N is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERRS_N(i,:) corresponds to the ith right-hand side.

The second index in ERRS_N(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

This subroutine is only responsible for setting the second field above. See Lapack Working Note 165 for further details and extra cautions.

ERRS_C Input and output parameter.

ERRS_C is DOUBLE PRECISION

ERRS_C is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERRS_C is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:,N_ERR_BNDS) entries are returned.

The first index in ERRS_C(i,:) corresponds to the ith right-hand side.

The second index in ERRS_C(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

This subroutine is only responsible for setting the second field above. See Lapack Working Note 165 for further details and extra cautions.

RES Input parameter.

RES is COMPLEX*16

RES is an array, dimension (N). Workspace to hold the intermediate residual.

AYB Input parameter.

AYB is DOUBLE PRECISION

AYB is an array, dimension (N). Workspace.

DY Input parameter.

DY is COMPLEX*16

DY is an array, dimension (N). Workspace to hold the intermediate solution.

Y_TAIL Input parameter.

Y_TAIL is COMPLEX*16

Y_TAIL is an array, dimension (N). Workspace to hold the trailing bits of the intermediate solution.

RCOND Input parameter.

RCOND is DOUBLE PRECISION

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

ITHRESH Input parameter.

ITHRESH is INTEGER

The maximum number of residual computations allowed for refinement. The default is 10. For 'aggressive' set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in ERRS_N and ERRS_C may no longer be trustworthy.

RTHRESH Input parameter.

RTHRESH is DOUBLE PRECISION

Determines when to stop refinement if the error estimate stops decreasing. Refinement will stop when the next solution no longer satisfies norm(dx_{i+1}) < RTHRESH * norm(dx_i) where norm(Z) is the infinity norm of Z. RTHRESH satisfies 0 < RTHRESH <= 1. The default value is 0.5. For 'aggressive' set to 0.9 to permit convergence on extremely ill-conditioned matrices. See LAWN 165 for more details.

DZ_UB Input parameter.

DZ_UB is DOUBLE PRECISION

Determines when to start considering componentwise convergence. Componentwise convergence is only considered after each component of the solution Y is stable, which we definte as the relative change in each component being less than DZ_UB. The default value is 0.25, requiring the first bit to be stable. See LAWN 165 for more details.

IGNORE_CWISE Input parameter.

IGNORE_CWISE is LOGICAL

If .TRUE. then ignore componentwise convergence. Default value is .FALSE..

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. < 0: if INFO = -i, the ith argument to ZGETRS had an illegal value

Related Information

For this routine in other precisions, please see *cla_gerfsx_extended*, *dla_gerfsx_extended* and *sla_gerfsx_extended*.

4.17.660 zla_gerpvgrw

ZLA_GERPVGRW computes the reciprocal pivot growth factor norm(A)/norm(U). The “max absolute element” norm is used. If this is much less than 1, the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, estimated condition numbers, and error bounds could be unreliable.

Syntax

Fortran specification:

```
use armpl_library

double precision function zla_gerpvgrw(N, NCOLS, A, LDA, AF, LDAF)
```

C specification:

```
#include "armpl.h"

double zla_gerpvgrw_(const armpl_int_t *n, const armpl_int_t *ncols,
                     const armpl_doublecomplex_t *a, const armpl_int_t *lda,
                     const armpl_doublecomplex_t *af,
                     const armpl_int_t *ldaf);
```

Parameters

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NCOLS Input parameter.

NCOLS is INTEGER

The number of columns of the matrix A. NCOLS >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX*16

AF is an array, dimension (LDAF, N). The factors L and U from the factorization A = P*L*U as computed by ZGETRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

Related Information

For this routine in other precisions, please see *cla_gerpvgrw*, *dla_gerpvgrw* and *sla_gerpvgrw*.

4.17.661 zla_heamv

ZLA_SYAMV performs the matrix-vector operation

```
y := alpha*abs(A)*abs(x) + beta*abs(y),
```

where alpha and beta are scalars, x and y are vectors and A is an n by n symmetric matrix.

This function is primarily used in calculating error bounds. To protect against underflow during evaluation, components in the resulting vector are perturbed away from zero by (N+1) times the underflow threshold. To prevent unnecessarily large errors for block-structure embedded in general matrices, "symbolically" zero components are not perturbed. A zero entry is considered "symbolic" if all multiplications involved in computing that entry have at least one zero multiplicand.

Syntax

Fortran specification:

```
use armpl_library

subroutine zla_heamv(UPLO, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
```

C specification:

```
#include "armpl.h"

void zla_heamv_(const armpl_int_t *uplo, const armpl_int_t *n,
                const double *alpha, const armpl_doublecomplex_t *a,
                const armpl_int_t *lda, const armpl_doublecomplex_t *x,
                const armpl_int_t *incx, const double *beta, double *y,
                const armpl_int_t *incy);
```

Parameters

UPLO Input parameter.

UPLO is INTEGER

On entry, UPLO specifies whether the upper or lower triangular part of the array A is to be referenced as follows:

UPLO = BLAS_UPPER Only the upper triangular part of A is to be referenced.

UPLO = BLAS_LOWER Only the lower triangular part of A is to be referenced.

Unchanged on exit.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix A. N must be at least zero. Unchanged on exit.

ALPHA Input parameter.

ALPHA is DOUBLE PRECISION .

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, n).. Before entry, the leading m by n part of the array A must contain the matrix of coefficients. Unchanged on exit.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, n). Unchanged on exit.

X Input parameter.

X is COMPLEX*16

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)) Before entry, the incremented array X must contain the vector x. Unchanged on exit.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

BETA Input parameter.

BETA is DOUBLE PRECISION .

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input. Unchanged on exit.

Y Input and output parameter.

Y is DOUBLE PRECISION

Y is an array, dimension. (1 + (n - 1)*abs(INCY)) Before entry with BETA non-zero, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero. Unchanged on exit.

Related Information

For this routine in other precisions, please see *cla_heamv*.

4.17.662 zla_hercond_c

```
ZLA_HERCOND_C computes the infinity norm condition number of
   op(A) * inv(diag(C)) where C is a DOUBLE PRECISION vector.
```

Syntax

Fortran specification:

```
use armpl_library

double precision function zla_hercond_c(UPLO, N, A, LDA, AF, LDAF, IPIV, C,
                                        CAPPLY, INFO, WORK, RWORK)
```

C specification:

```
#include "armpl.h"

double zla_hercond_c_(const char *uplo, const armpl_int_t *n,
                      const armpl_doublecomplex_t *a, const armpl_int_t *lda,
```

(continues on next page)

(continued from previous page)

```
                        const armpl_doublecomplex_t *af,
                        const armpl_int_t *ldaf, const armpl_int_t *ipiv,
                        const double *c, const armpl_int_t *capply,
                        armpl_int_t *info, const armpl_doublecomplex_t *work,
                        const double *rwork, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX*16

AF is an array, dimension (LDAF, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by ZHETRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CHETRF.

C Input parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). The vector C in the formula op(A) * inv(diag(C)).

CAPPLY Input parameter.

CAPPLY is LOGICAL

If .TRUE. then access the vector C in the formula above.

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. i > 0: The ith argument is invalid.

WORK Input parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N).. Workspace.

RWORK Input parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N).. Workspace.

Related Information

For this routine in other precisions, please see *cla_hercond_c*.

4.17.663 zla_hercond_x

```
ZLA_HERCOND_X computes the infinity norm condition number of
   op(A) * diag(X) where X is a COMPLEX*16 vector.
```

Syntax

Fortran specification:

```
use armpl_library

double precision function zla_hercond_x(UPLO, N, A, LDA, AF, LDAF, IPIV, X,
                                        INFO, WORK, RWORK)
```

C specification:

```
#include "armpl.h"

double zla_hercond_x_(const char *uplo, const armpl_int_t *n,
                      const armpl_doublecomplex_t *a, const armpl_int_t *lda,
                      const armpl_doublecomplex_t *af,
                      const armpl_int_t *ldaf, const armpl_int_t *ipiv,
                      const armpl_doublecomplex_t *x, armpl_int_t *info,
                      const armpl_doublecomplex_t *work, const double *rwork,
                      ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX*16

AF is an array, dimension (LDAF, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by ZHETRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by CHETRF.

X Input parameter.

X is COMPLEX*16

X is an array, dimension (N). The vector X in the formula op(A) * diag(X).

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. i > 0: The ith argument is invalid.

WORK Input parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N).. Workspace.

RWORK Input parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N).. Workspace.

Related Information

For this routine in other precisions, please see *cla_hercond_x*.

4.17.664 zla_herfsx_extended

zla_herfsx_extended improves the computed solution to a system of linear equations by performing extra-precise iterative refinement and provides error bounds and backward error estimates for the solution. This subroutine is called by ZHERFSX to perform iterative refinement. In addition to normwise error bound, the code provides maximum componentwise error bound if possible. See comments for ERR_BNDS_NORM and ERR_BNDS_COMP for details of the error bounds. Note that this subroutine is only resonsible for setting the second fields of ERR_BNDS_NORM and ERR_BNDS_COMP.

Syntax

Fortran specification:

```
use armpl_library

subroutine zla_herfsx_extended(PREC_TYPE, UPLO, N, NRHS, A, LDA, AF, LDAF,
                               IPIV, COLEQU, C, B, LDB, Y, LDY, BERR_OUT,
                               N_NORMS, ERR_BNDS_NORM, ERR_BNDS_COMP, RES,
                               AYB, DY, Y_TAIL, RCOND, ITHRESH, RTHRESH,
                               DZ_UB, IGNORE_CWISE, INFO)
```

C specification:

```
#include "armpl.h"

void zla_herfsx_extended_(const armpl_int_t *prec_type, const char *uplo,
                          const armpl_int_t *n, const armpl_int_t *nrhs,
                          const armpl_doublecomplex_t *a,
                          const armpl_int_t *lda,
                          const armpl_doublecomplex_t *af,
                          const armpl_int_t *ldaf, const armpl_int_t *ipiv,
                          const armpl_int_t *colequ, const double *c,
                          const armpl_doublecomplex_t *b,
                          const armpl_int_t *ldb, armpl_doublecomplex_t *y,
                          const armpl_int_t *ldy, double *berr_out,
                          const armpl_int_t *n_norms, double *err_bnds_norm,
                          double *err_bnds_comp, armpl_doublecomplex_t *res,
                          double *ayb, armpl_doublecomplex_t *dy,
                          armpl_doublecomplex_t *y_tail, const double *rcond,
                          const armpl_int_t *ithresh, const double *rthresh,
                          const double *dz_ub,
                          const armpl_int_t *ignore_cwise, armpl_int_t *info,
                          ... );
```

Parameters

PREC_TYPE Input parameter.

PREC_TYPE is INTEGER

Specifies the intermediate precision to be used in refinement. The value is defined by ILAPREC(P) where P is a CHARACTER and P = 'S': Single = 'D': Double = 'I': Indigenous = 'X', 'E': Extra

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right-hand-sides, i.e., the number of columns of the matrix B.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX*16

AF is an array, dimension (LDAF, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by ZHETRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZHETRF.

COLEQU Input parameter.

COLEQU is LOGICAL

If .TRUE. then column equilibration was done to A before calling this routine. This is needed to compute the solution and error bounds correctly.

C Input parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). The column scale factors for A. If COLEQU = .FALSE., C is not accessed. If C is input, each element of C should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). The right-hand-side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Y Input and output parameter.

Y is COMPLEX*16

Y is an array, dimension (LDY, NRHS). On entry, the solution matrix X, as computed by ZHETRS. On exit, the improved solution matrix Y.

LDY Input parameter.

LDY is INTEGER

The leading dimension of the array Y. LDY >= max(1, N).

BERR_OUT Output parameter.

BERR_OUT is DOUBLE PRECISION

BERR_OUT is an array, dimension (NRHS). On exit, BERR_OUT(j) contains the componentwise relative backward error for right-hand-side j from the formula max(i) (abs(RES(i)) / (abs(op(A_s))*abs(Y) + abs(B_s))(i)) where abs(Z) is the componentwise absolute value of the matrix or vector Z. This is computed by ZLA_LIN_BERR.

N_NORMS Input parameter.

N_NORMS is INTEGER

Determines which error bounds to return (see ERR_BNDS_NORM and ERR_BNDS_COMP). If N_NORMS >= 1 return normwise error bounds. If N_NORMS >= 2 return componentwise error bounds.

ERR_BNDS_NORM Input and output parameter.

ERR_BNDS_NORM is DOUBLE PRECISION

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

This subroutine is only responsible for setting the second field above. See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Input and output parameter.

ERR_BNDS_COMP is DOUBLE PRECISION

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:,N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix

Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

This subroutine is only responsible for setting the second field above. See Lapack Working Note 165 for further details and extra cautions.

RES Input parameter.

RES is COMPLEX*16

RES is an array, dimension (N). Workspace to hold the intermediate residual.

AYB Input parameter.

AYB is DOUBLE PRECISION

AYB is an array, dimension (N). Workspace.

DY Input parameter.

DY is COMPLEX*16

DY is an array, dimension (N). Workspace to hold the intermediate solution.

Y_TAIL Input parameter.

Y_TAIL is COMPLEX*16

Y_TAIL is an array, dimension (N). Workspace to hold the trailing bits of the intermediate solution.

RCOND Input parameter.

RCOND is DOUBLE PRECISION

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

ITHRESH Input parameter.

ITHRESH is INTEGER

The maximum number of residual computations allowed for refinement. The default is 10. For 'aggressive' set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in ERR_BNDS_NORM and ERR_BNDS_COMP may no longer be trustworthy.

RTHRESH Input parameter.

RTHRESH is DOUBLE PRECISION

Determines when to stop refinement if the error estimate stops decreasing. Refinement will stop when the next solution no longer satisfies norm(dx_{i+1}) < RTHRESH * norm(dx_i) where norm(Z) is the infinity norm of Z. RTHRESH satisfies 0 < RTHRESH <= 1. The default value is 0.5. For 'aggressive' set to 0.9 to permit convergence on extremely ill-conditioned matrices. See LAWN 165 for more details.

DZ_UB Input parameter.

DZ_UB is DOUBLE PRECISION

Determines when to start considering componentwise convergence. Componentwise convergence is only considered after each component of the solution Y is stable, which we definte as the relative change in each component being less than DZ_UB. The default value is 0.25, requiring the first bit to be stable. See LAWN 165 for more details.

IGNORE_CWISE Input parameter.

IGNORE_CWISE is LOGICAL

If .TRUE. then ignore componentwise convergence. Default value is .FALSE..

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. < 0: if INFO = -i, the ith argument to ZLA_HERFSX_EXTENDED had an illegal value

Related Information

For this routine in other precisions, please see *cla_herfsx_extended*.

4.17.665 zla_herpvgrw

ZLA_HERPVGRW computes the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If this is much less than 1, the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, estimated condition numbers, and error bounds could be unreliable.

Syntax

Fortran specification:

```
use armpl_library

double precision function zla_herpvgrw(UPLO, N, INFO, A, LDA, AF, LDAF, IPIV,
                                       WORK)
```

C specification:

```
#include "armpl.h"

double zla_herpvgrw_(const char *uplo, const armpl_int_t *n,
                     const armpl_int_t *info, const armpl_doublecomplex_t *a,
                     const armpl_int_t *lda, const armpl_doublecomplex_t *af,
                     const armpl_int_t *ldaf, const armpl_int_t *ipiv,
                     double *work, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

INFO Input parameter.

INFO is INTEGER

The value of INFO returned from ZHETRF, .i.e., the pivot in column INFO is exactly 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX*16

AF is an array, dimension (LDAF, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by ZHETRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZHETRF.

WORK Input parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (2*N) .

Related Information

For this routine in other precisions, please see *cla_herpvgrw*.

4.17.666 zla_lin_berr

```
ZLA_LIN_BERR computes componentwise relative backward error from
   the formula
       max(i) ( abs(R(i)) / ( abs(op(A_s))*abs(Y) + abs(B_s) )(i) )
   where abs(Z) is the componentwise absolute value of the matrix
   or vector Z.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zla_lin_berr(N, NZ, NRHS, RES, AYB, BERR)
```

C specification:

```
#include "armpl.h"

void zla_lin_berr_(const armpl_int_t *n, const armpl_int_t *nz,
                   const armpl_int_t *nrhs, const armpl_doublecomplex_t *res,
                   const double *ayb, double *berr);
```

Parameters

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NZ Input parameter.

NZ is INTEGER

We add (NZ+1)*SLAMCH('Safe minimum') to R(i) in the numerator to guard against spuriously zero residuals. Default value is N.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices AYB, RES, and BERR. NRHS >= 0.

RES Input parameter.

RES is COMPLEX*16

RES is an array, dimension (N, NRHS). The residual matrix, i.e., the matrix R in the relative backward error formula above.

AYB Input parameter.

AYB is DOUBLE PRECISION

AYB is an array, dimension (N, NRHS). The denominator in the relative backward error formula above, i.e., the matrix abs(op(A_s))*abs(Y) + abs(B_s). The matrices A, Y, and B are from iterative refinement (see zla_gerfsx_extended.f).

BERR Output parameter.

BERR is DOUBLE PRECISION

BERR is an array, dimension (NRHS). The componentwise relative backward error from the formula above.

Related Information

For this routine in other precisions, please see *cla_lin_berr*, *dla_lin_berr* and *sla_lin_berr*.

4.17.667 zla_porcond_c

```
ZLA_PORCOND_C Computes the infinity norm condition number of
   op(A) * inv(diag(C)) where C is a DOUBLE PRECISION vector
```

Syntax

Fortran specification:

```
use armpl_library

double precision function zla_porcond_c(UPLO, N, A, LDA, AF, LDAF, C, CAPPLY,
                                        INFO, WORK, RWORK)
```

C specification:

```
#include "armpl.h"

double zla_porcond_c_(const char *uplo, const armpl_int_t *n,
                      const armpl_doublecomplex_t *a, const armpl_int_t *lda,
                      const armpl_doublecomplex_t *af,
                      const armpl_int_t *ldaf, const double *c,
                      const armpl_int_t *capply, armpl_int_t *info,
                      const armpl_doublecomplex_t *work, const double *rwork,
                      ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX*16

AF is an array, dimension (LDAF, N). The triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H , as computed by ZPOTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

C Input parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). The vector C in the formula op(A) * inv(diag(C)).

CAPPLY Input parameter.

CAPPLY is LOGICAL

If .TRUE. then access the vector C in the formula above.

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. i > 0: The ith argument is invalid.

WORK Input parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N).. Workspace.

RWORK Input parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N).. Workspace.

Related Information

For this routine in other precisions, please see *cla_porcond_c*.

4.17.668 zla_porcond_x

```
ZLA_PORCOND_X Computes the infinity norm condition number of
   op(A) * diag(X) where X is a COMPLEX*16 vector.
```

Syntax

Fortran specification:

```
use armpl_library

double precision function zla_porcond_x(UPLO, N, A, LDA, AF, LDAF, X, INFO,
                                        WORK, RWORK)
```

C specification:

```
#include "armpl.h"

double zla_porcond_x_(const char *uplo, const armpl_int_t *n,
                      const armpl_doublecomplex_t *a, const armpl_int_t *lda,
                      const armpl_doublecomplex_t *af,
                      const armpl_int_t *ldaf, const armpl_doublecomplex_t *x,
                      armpl_int_t *info, const armpl_doublecomplex_t *work,
                      const double *rwork, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX*16

AF is an array, dimension (LDAF, N). The triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H , as computed by ZPOTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

X Input parameter.

X is COMPLEX*16

X is an array, dimension (N). The vector X in the formula op(A) * diag(X).

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. i > 0: The ith argument is invalid.

WORK Input parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N).. Workspace.

RWORK Input parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N).. Workspace.

Related Information

For this routine in other precisions, please see *cla_porcond_x*.

4.17.669 zla_porfsx_extended

zla_porfsx_extended improves the computed solution to a system of linear equations by performing extra-precise iterative refinement and provides error bounds and backward error estimates for the solution. This subroutine is called by ZPORFSX to perform iterative refinement. In addition to normwise error bound, the code provides maximum componentwise error bound if possible. See comments for ERR_BNDS_NORM and ERR_BNDS_COMP for details of the error bounds. Note that this subroutine is only resonsible for setting the second fields of ERR_BNDS_NORM and ERR_BNDS_COMP.

Syntax

Fortran specification:

```
use armpl_library

subroutine zla_porfsx_extended(PREC_TYPE, UPLO, N, NRHS, A, LDA, AF, LDAF,
                               COLEQU, C, B, LDB, Y, LDY, BERR_OUT, N_NORMS,
                               ERR_BNDS_NORM, ERR_BNDS_COMP, RES, AYB, DY,
                               Y_TAIL, RCOND, ITHRESH, RTHRESH, DZ_UB,
                               IGNORE_CWISE, INFO)
```

C specification:

```
#include "armpl.h"

void zla_porfsx_extended_(const armpl_int_t *prec_type, const char *uplo,
                          const armpl_int_t *n, const armpl_int_t *nrhs,
                          const armpl_doublecomplex_t *a,
                          const armpl_int_t *lda,
                          const armpl_doublecomplex_t *af,
                          const armpl_int_t *ldaf, const armpl_int_t *colequ,
                          const double *c, const armpl_doublecomplex_t *b,
                          const armpl_int_t *ldb, armpl_doublecomplex_t *y,
                          const armpl_int_t *ldy, double *berr_out,
                          const armpl_int_t *n_norms, double *err_bnds_norm,
                          double *err_bnds_comp, armpl_doublecomplex_t *res,
                          double *ayb, armpl_doublecomplex_t *dy,
                          armpl_doublecomplex_t *y_tail, const double *rcond,
                          const armpl_int_t *ithresh, const double *rthresh,
                          const double *dz_ub,
                          const armpl_int_t *ignore_cwise, armpl_int_t *info,
                          ... );
```

Parameters

PREC_TYPE Input parameter.

PREC_TYPE is INTEGER

Specifies the intermediate precision to be used in refinement. The value is defined by ILAPREC(P) where P is a CHARACTER and P = 'S': Single = 'D': Double = 'I': Indigenous = 'X', 'E': Extra

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right-hand-sides, i.e., the number of columns of the matrix B.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX*16

AF is an array, dimension (LDAF, N). The triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , as computed by ZPOTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

COLEQU Input parameter.

COLEQU is LOGICAL

If .TRUE. then column equilibration was done to A before calling this routine. This is needed to compute the solution and error bounds correctly.

C Input parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). The column scale factors for A. If COLEQU = .FALSE., C is not accessed. If C is input, each element of C should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). The right-hand-side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Y Input and output parameter.

Y is COMPLEX*16

Y is an array, dimension (LDY, NRHS). On entry, the solution matrix X, as computed by ZPOTRS. On exit, the improved solution matrix Y.

LDY Input parameter.

LDY is INTEGER

The leading dimension of the array Y. LDY >= max(1, N).

BERR_OUT Output parameter.

BERR_OUT is DOUBLE PRECISION

BERR_OUT is an array, dimension (NRHS). On exit, BERR_OUT(j) contains the componentwise relative backward error for right-hand-side j from the formula max(i) (abs(RES(i)) / (abs(op(A_s))*abs(Y) + abs(B_s))(i)) where abs(Z) is the componentwise absolute value of the matrix or vector Z. This is computed by ZLA_LIN_BERR.

N_NORMS Input parameter.

N_NORMS is INTEGER

Determines which error bounds to return (see ERR_BNDS_NORM and ERR_BNDS_COMP). If N_NORMS >= 1 return normwise error bounds. If N_NORMS >= 2 return componentwise error bounds.

ERR_BNDS_NORM Input and output parameter.

ERR_BNDS_NORM is DOUBLE PRECISION

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

This subroutine is only responsible for setting the second field above. See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Input and output parameter.

ERR_BNDS_COMP is DOUBLE PRECISION

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:,N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

This subroutine is only responsible for setting the second field above. See Lapack Working Note 165 for further details and extra cautions.

RES Input parameter.

RES is COMPLEX*16

RES is an array, dimension (N). Workspace to hold the intermediate residual.

AYB Input parameter.

AYB is DOUBLE PRECISION

AYB is an array, dimension (N). Workspace.

DY Input parameter.

DY is COMPLEX*16 PRECISION array, dimension (N)

Workspace to hold the intermediate solution.

Y_TAIL Input parameter.

Y_TAIL is COMPLEX*16

Y_TAIL is an array, dimension (N). Workspace to hold the trailing bits of the intermediate solution.

RCOND Input parameter.

RCOND is DOUBLE PRECISION

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

ITHRESH Input parameter.

ITHRESH is INTEGER

The maximum number of residual computations allowed for refinement. The default is 10. For 'aggressive' set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in ERR_BNDS_NORM and ERR_BNDS_COMP may no longer be trustworthy.

RTHRESH Input parameter.

RTHRESH is DOUBLE PRECISION

Determines when to stop refinement if the error estimate stops decreasing. Refinement will stop when the next solution no longer satisfies norm(dx_{i+1}) < RTHRESH * norm(dx_i) where norm(Z) is the infinity norm of Z. RTHRESH satisfies 0 < RTHRESH <= 1. The default value is 0.5. For 'aggressive' set to 0.9 to permit convergence on extremely ill-conditioned matrices. See LAWN 165 for more details.

DZ_UB Input parameter.

DZ_UB is DOUBLE PRECISION

Determines when to start considering componentwise convergence. Componentwise convergence is only considered after each component of the solution Y is stable, which we definte as the relative change in each component being less than DZ_UB. The default value is 0.25, requiring the first bit to be stable. See LAWN 165 for more details.

IGNORE_CWISE Input parameter.

IGNORE_CWISE is LOGICAL

If .TRUE. then ignore componentwise convergence. Default value is .FALSE..

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. < 0: if INFO = -i, the ith argument to ZPOTRS had an illegal value

Related Information

For this routine in other precisions, please see *cla_porfsx_extended*, *dla_porfsx_extended* and *sla_porfsx_extended*.

4.17.670 zla_porpvgrw

ZLA_PORPVGRW computes the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If this is much less than 1, the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, estimated condition numbers, and error bounds could be unreliable.

Syntax

Fortran specification:

```
use armpl_library

double precision function zla_porpvgrw(UPLO, NCOLS, A, LDA, AF, LDAF, WORK)
```

C specification:

```
#include "armpl.h"

double zla_porpvgrw_(const char *uplo, const armpl_int_t *ncols,
                     const armpl_doublecomplex_t *a, const armpl_int_t *lda,
                     const armpl_doublecomplex_t *af, const armpl_int_t *ldaf,
                     double *work, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

NCOLS Input parameter.

NCOLS is INTEGER

The number of columns of the matrix A. NCOLS >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA,N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1,N).

AF Input parameter.

AF is COMPLEX*16

AF is an array, dimension (LDAF,N). The triangular factor U or L from the Cholesky factorization A = U^T *U or A = L*L^T , as computed by ZPOTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1,N).

WORK Input parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (2*N) .

Related Information

For this routine in other precisions, please see *cla_porpvgrw*, *dla_porpvgrw* and *sla_porpvgrw*.

4.17.671 zla_syamv

ZLA_SYAMV performs the matrix-vector operation

```
y := alpha*abs(A)*abs(x) + beta*abs(y),
```

where alpha and beta are scalars, x and y are vectors and A is an n by n symmetric matrix.

This function is primarily used in calculating error bounds. To protect against underflow during evaluation, components in the resulting vector are perturbed away from zero by (N+1) times the underflow threshold. To prevent unnecessarily large errors for block-structure embedded in general matrices, "symbolically" zero components are not perturbed. A zero entry is considered "symbolic" if all multiplications involved in computing that entry have at least one zero multiplicand.

Syntax

Fortran specification:

```
use armpl_library

subroutine zla_syamv(UPLO, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
```

C specification:

```
#include "armpl.h"

void zla_syamv_(const armpl_int_t *uplo, const armpl_int_t *n,
                const double *alpha, const armpl_doublecomplex_t *a,
                const armpl_int_t *lda, const armpl_doublecomplex_t *x,
                const armpl_int_t *incx, const double *beta, double *y,
                const armpl_int_t *incy);
```

Parameters

UPLO Input parameter.

UPLO is INTEGER

On entry, UPLO specifies whether the upper or lower triangular part of the array A is to be referenced as follows:

UPLO = BLAS_UPPER Only the upper triangular part of A is to be referenced.

UPLO = BLAS_LOWER Only the lower triangular part of A is to be referenced.

Unchanged on exit.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of the matrix A. N must be at least zero. Unchanged on exit.

ALPHA Input parameter.

ALPHA is DOUBLE PRECISION .

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, n).. Before entry, the leading m by n part of the array A must contain the matrix of coefficients. Unchanged on exit.

LDA Input parameter.

LDA is INTEGER

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, n). Unchanged on exit.

X Input parameter.

X is COMPLEX*16

X is an array, dimension at least. (1 + (n - 1)*abs(INCX)) Before entry, the incremented array X must contain the vector x. Unchanged on exit.

INCX Input parameter.

INCX is INTEGER

On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

BETA Input parameter.

BETA is DOUBLE PRECISION .

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input. Unchanged on exit.

Y Input and output parameter.

Y is DOUBLE PRECISION

Y is an array, dimension. (1 + (n - 1)*abs(INCY)) Before entry with BETA non-zero, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.

INCY Input parameter.

INCY is INTEGER

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero. Unchanged on exit.

Related Information

For this routine in other precisions, please see *cla_syamv*, *dla_syamv* and *sla_syamv*.

4.17.672 zla_syrcond_c

```
ZLA_SYRCOND_C Computes the infinity norm condition number of
   op(A) * inv(diag(C)) where C is a DOUBLE PRECISION vector.
```

Syntax

Fortran specification:

```
use armpl_library

double precision function zla_syrcond_c(UPLO, N, A, LDA, AF, LDAF, IPIV, C,
                                        CAPPLY, INFO, WORK, RWORK)
```

C specification:

```
#include "armpl.h"

double zla_syrcond_c_(const char *uplo, const armpl_int_t *n,
                      const armpl_doublecomplex_t *a, const armpl_int_t *lda,
```

(continues on next page)

(continued from previous page)

```
                        const armpl_doublecomplex_t *af,
                        const armpl_int_t *ldaf, const armpl_int_t *ipiv,
                        const double *c, const armpl_int_t *capply,
                        armpl_int_t *info, const armpl_doublecomplex_t *work,
                        const double *rwork, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX*16

AF is an array, dimension (LDAF, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by ZSYTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZSYTRF.

C Input parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). The vector C in the formula op(A) * inv(diag(C)).

CAPPLY Input parameter.

CAPPLY is LOGICAL

If .TRUE. then access the vector C in the formula above.

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. i > 0: The ith argument is invalid.

WORK Input parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N).. Workspace.

RWORK Input parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N).. Workspace.

Related Information

For this routine in other precisions, please see *cla_syrcond_c*.

4.17.673 zla_syrcond_x

```
ZLA_SYRCOND_X Computes the infinity norm condition number of
   op(A) * diag(X) where X is a COMPLEX*16 vector.
```

Syntax

Fortran specification:

```
use armpl_library

double precision function zla_syrcond_x(UPLO, N, A, LDA, AF, LDAF, IPIV, X,
                                        INFO, WORK, RWORK)
```

C specification:

```
#include "armpl.h"

double zla_syrcond_x_(const char *uplo, const armpl_int_t *n,
                      const armpl_doublecomplex_t *a, const armpl_int_t *lda,
                      const armpl_doublecomplex_t *af,
                      const armpl_int_t *ldaf, const armpl_int_t *ipiv,
                      const armpl_doublecomplex_t *x, armpl_int_t *info,
                      const armpl_doublecomplex_t *work, const double *rwork,
                      ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX*16

AF is an array, dimension (LDAF, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by ZSYTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZSYTRF.

X Input parameter.

X is COMPLEX*16

X is an array, dimension (N). The vector X in the formula op(A) * diag(X).

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. i > 0: The ith argument is invalid.

WORK Input parameter.

WORK is COMPLEX*16

WORK is an array, dimension (2*N).. Workspace.

RWORK Input parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N).. Workspace.

Related Information

For this routine in other precisions, please see *cla_syrcond_x*.

4.17.674 zla_syrfsx_extended

zla_syrfsx_extended improves the computed solution to a system of linear equations by performing extra-precise iterative refinement and provides error bounds and backward error estimates for the solution. This subroutine is called by ZSYRFSX to perform iterative refinement. In addition to normwise error bound, the code provides maximum componentwise error bound if possible. See comments for ERR_BNDS_NORM and ERR_BNDS_COMP for details of the error bounds. Note that this subroutine is only resonsible for setting the second fields of ERR_BNDS_NORM and ERR_BNDS_COMP.

Syntax

Fortran specification:

```
use armpl_library

subroutine zla_syrfsx_extended(PREC_TYPE, UPLO, N, NRHS, A, LDA, AF, LDAF,
                               IPIV, COLEQU, C, B, LDB, Y, LDY, BERR_OUT,
                               N_NORMS, ERR_BNDS_NORM, ERR_BNDS_COMP, RES,
                               AYB, DY, Y_TAIL, RCOND, ITHRESH, RTHRESH,
                               DZ_UB, IGNORE_CWISE, INFO)
```

C specification:

```
#include "armpl.h"

void zla_syrfsx_extended_(const armpl_int_t *prec_type, const char *uplo,
                          const armpl_int_t *n, const armpl_int_t *nrhs,
                          const armpl_doublecomplex_t *a,
                          const armpl_int_t *lda,
                          const armpl_doublecomplex_t *af,
                          const armpl_int_t *ldaf, const armpl_int_t *ipiv,
                          const armpl_int_t *colequ, const double *c,
                          const armpl_doublecomplex_t *b,
                          const armpl_int_t *ldb, armpl_doublecomplex_t *y,
                          const armpl_int_t *ldy, double *berr_out,
                          const armpl_int_t *n_norms, double *err_bnds_norm,
                          double *err_bnds_comp, armpl_doublecomplex_t *res,
                          double *ayb, armpl_doublecomplex_t *dy,
                          armpl_doublecomplex_t *y_tail, const double *rcond,
                          const armpl_int_t *ithresh, const double *rthresh,
                          const double *dz_ub,
                          const armpl_int_t *ignore_cwise, armpl_int_t *info,
                          ... );
```

Parameters

PREC_TYPE Input parameter.

PREC_TYPE is INTEGER

Specifies the intermediate precision to be used in refinement. The value is defined by ILAPREC(P) where P is a CHARACTER and P = 'S': Single = 'D': Double = 'I': Indigenous = 'X', 'E': Extra

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right-hand-sides, i.e., the number of columns of the matrix B.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX*16

AF is an array, dimension (LDAF, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by ZSYTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZSYTRF.

COLEQU Input parameter.

COLEQU is LOGICAL

If .TRUE. then column equilibration was done to A before calling this routine. This is needed to compute the solution and error bounds correctly.

C Input parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). The column scale factors for A. If COLEQU = .FALSE., C is not accessed. If C is input, each element of C should be a power of the radix to ensure a reliable solution and error estimates. Scaling by powers of the radix does not cause rounding errors unless the result underflows or overflows. Rounding errors during scaling lead to refining with a matrix that is not equivalent to the input matrix, producing error estimates that may not be reliable.

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). The right-hand-side matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Y Input and output parameter.

Y is COMPLEX*16

Y is an array, dimension (LDY, NRHS). On entry, the solution matrix X, as computed by ZSYTRS. On exit, the improved solution matrix Y.

LDY Input parameter.

LDY is INTEGER

The leading dimension of the array Y. LDY >= max(1, N).

BERR_OUT Output parameter.

BERR_OUT is DOUBLE PRECISION

BERR_OUT is an array, dimension (NRHS). On exit, BERR_OUT(j) contains the componentwise relative backward error for right-hand-side j from the formula max(i) (abs(RES(i)) / (abs(op(A_s))*abs(Y) + abs(B_s))(i)) where abs(Z) is the componentwise absolute value of the matrix or vector Z. This is computed by ZLA_LIN_BERR.

N_NORMS Input parameter.

N_NORMS is INTEGER

Determines which error bounds to return (see ERR_BNDS_NORM and ERR_BNDS_COMP). If N_NORMS >= 1 return normwise error bounds. If N_NORMS >= 2 return componentwise error bounds.

ERR_BNDS_NORM Input and output parameter.

ERR_BNDS_NORM is DOUBLE PRECISION

ERR_BNDS_NORM is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the normwise relative error, which is defined as follows:

Normwise relative error in the ith solution vector: max_j (abs(XTRUE(j,i) - X(j,i))) ——————————— max_j abs(X(j,i))

The array is indexed by the type of error information as described below. There currently are up to three pieces of information returned.

The first index in ERR_BNDS_NORM(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_NORM(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated normwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix Z. Let Z = S*A, where S scales each row by a power of the radix so all absolute row sums of Z are approximately 1.

This subroutine is only responsible for setting the second field above. See Lapack Working Note 165 for further details and extra cautions.

ERR_BNDS_COMP Input and output parameter.

ERR_BNDS_COMP is DOUBLE PRECISION

ERR_BNDS_COMP is an array, dimension (NRHS, N_ERR_BNDS). For each right-hand side, this array contains information about various error bounds and condition numbers corresponding to the componentwise relative error, which is defined as follows:

Componentwise relative error in the ith solution vector: abs(XTRUE(j,i) - X(j,i)) max_j ——————————- abs(X(j,i))

The array is indexed by the right-hand side i (on which the componentwise relative error depends), and the type of error information as described below. There currently are up to three pieces of information returned for each right-hand side. If componentwise accuracy is not requested (PARAMS(3) = 0.0), then ERR_BNDS_COMP is not accessed. If N_ERR_BNDS .LT. 3, then at most the first (:,N_ERR_BNDS) entries are returned.

The first index in ERR_BNDS_COMP(i,:) corresponds to the ith right-hand side.

The second index in ERR_BNDS_COMP(:,err) contains the following three fields: err = 1 "Trust/don't trust" boolean. Trust the answer if the reciprocal condition number is less than the threshold sqrt(n) * slamch('Epsilon').

err = 2 "Guaranteed" error bound: The estimated forward error, almost certainly within a factor of 10 of the true error so long as the next entry is greater than the threshold sqrt(n) * slamch('Epsilon'). This error bound should only be trusted if the previous boolean is true.

err = 3 Reciprocal condition number: Estimated componentwise reciprocal condition number. Compared with the threshold sqrt(n) * slamch('Epsilon') to determine if the error estimate is "guaranteed". These reciprocal condition numbers are 1 / (norm(Z^{-1},inf) * norm(Z,inf)) for some appropriately scaled matrix

Z. Let Z = S*(A*diag(x)), where x is the solution for the current right-hand side and S scales each row of A*diag(x) by a power of the radix so all absolute row sums of Z are approximately 1.

This subroutine is only responsible for setting the second field above. See Lapack Working Note 165 for further details and extra cautions.

RES Input parameter.

RES is COMPLEX*16

RES is an array, dimension (N). Workspace to hold the intermediate residual.

AYB Input parameter.

AYB is DOUBLE PRECISION

AYB is an array, dimension (N). Workspace.

DY Input parameter.

DY is COMPLEX*16

DY is an array, dimension (N). Workspace to hold the intermediate solution.

Y_TAIL Input parameter.

Y_TAIL is COMPLEX*16

Y_TAIL is an array, dimension (N). Workspace to hold the trailing bits of the intermediate solution.

RCOND Input parameter.

RCOND is DOUBLE PRECISION

Reciprocal scaled condition number. This is an estimate of the reciprocal Skeel condition number of the matrix A after equilibration (if done). If this is less than the machine precision (in particular, if it is zero), the matrix is singular to working precision. Note that the error may still be small even if this number is very small and the matrix appears ill- conditioned.

ITHRESH Input parameter.

ITHRESH is INTEGER

The maximum number of residual computations allowed for refinement. The default is 10. For 'aggressive' set to 100 to permit convergence using approximate factorizations or factorizations other than LU. If the factorization uses a technique other than Gaussian elimination, the guarantees in ERR_BNDS_NORM and ERR_BNDS_COMP may no longer be trustworthy.

RTHRESH Input parameter.

RTHRESH is DOUBLE PRECISION

Determines when to stop refinement if the error estimate stops decreasing. Refinement will stop when the next solution no longer satisfies norm(dx_{i+1}) < RTHRESH * norm(dx_i) where norm(Z) is the infinity norm of Z. RTHRESH satisfies 0 < RTHRESH <= 1. The default value is 0.5. For 'aggressive' set to 0.9 to permit convergence on extremely ill-conditioned matrices. See LAWN 165 for more details.

DZ_UB Input parameter.

DZ_UB is DOUBLE PRECISION

Determines when to start considering componentwise convergence. Componentwise convergence is only considered after each component of the solution Y is stable, which we definte as the relative change in each component being less than DZ_UB. The default value is 0.25, requiring the first bit to be stable. See LAWN 165 for more details.

IGNORE_CWISE Input parameter.

IGNORE_CWISE is LOGICAL

If .TRUE. then ignore componentwise convergence. Default value is .FALSE..

INFO Output parameter.

INFO is INTEGER

= 0: Successful exit. < 0: if INFO = -i, the ith argument to ZLA_HERFSX_EXTENDED had an illegal value

Related Information

For this routine in other precisions, please see *cla_syrfsx_extended*, *dla_syrfsx_extended* and *sla_syrfsx_extended*.

4.17.675 zla_syrpvgrw

ZLA_SYRPVGRW computes the reciprocal pivot growth factor norm(A)/norm(U). The "max absolute element" norm is used. If this is much less than 1, the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, estimated condition numbers, and error bounds could be unreliable.

Syntax

Fortran specification:

```
use armpl_library

double precision function zla_syrpvgrw(UPLO, N, INFO, A, LDA, AF, LDAF, IPIV,
                                       WORK)
```

C specification:

```
#include "armpl.h"

double zla_syrpvgrw_(const char *uplo, const armpl_int_t *n,
                     const armpl_int_t *info, const armpl_doublecomplex_t *a,
                     const armpl_int_t *lda, const armpl_doublecomplex_t *af,
                     const armpl_int_t *ldaf, const armpl_int_t *ipiv,
                     double *work, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

N Input parameter.

N is INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

INFO Input parameter.

INFO is INTEGER

The value of INFO returned from ZSYTRF, .i.e., the pivot in column INFO is exactly 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the N-by-N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AF Input parameter.

AF is COMPLEX*16

AF is an array, dimension (LDAF, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by ZSYTRF.

LDAF Input parameter.

LDAF is INTEGER

The leading dimension of the array AF. LDAF >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZSYTRF.

WORK Input parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (2*N) .

Related Information

For this routine in other precisions, please see *cla_syrpvgrw*, *dla_syrpvgrw* and *sla_syrpvgrw*.

4.17.676 zla_wwaddw

```
ZLA_WWADDW adds a vector W into a doubled-single vector (X, Y).

   This works for all extant IBM's hex and binary floating point
   arithmetics, but not for decimal.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zla_wwaddw(N, X, Y, W)
```

C specification:

```
#include "armpl.h"

void zla_wwaddw_(const armpl_int_t *n, armpl_doublecomplex_t *x,
                 armpl_doublecomplex_t *y, const armpl_doublecomplex_t *w);
```

Parameters

N Input parameter.

N is INTEGER

The length of vectors X, Y, and W.

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension (N). The first part of the doubled-single accumulation vector.

Y Input and output parameter.

Y is COMPLEX*16

Y is an array, dimension (N). The second part of the doubled-single accumulation vector.

W Input parameter.

W is COMPLEX*16

W is an array, dimension (N). The vector to be added.

Related Information

For this routine in other precisions, please see *cla_wwaddw*, *dla_wwaddw* and *sla_wwaddw*.

4.17.677 zlabrd

zlabrd reduces the first NB rows and columns of a complex general m by n matrix A to upper or lower real bidiagonal form by a unitary transformation Q^H * A * P, and returns the matrices X and Y which are needed to apply the transformation to the unreduced part of A.

If m >= n, A is reduced to upper bidiagonal form; if m < n, to lower bidiagonal form.

This is an auxiliary routine called by ZGEBRD

Syntax

Fortran specification:

```
use armpl_library

subroutine zlabrd(M, N, NB, A, LDA, D, E, TAUQ, TAUP, X, LDX, Y, LDY)
```

C specification:

```
#include "armpl.h"

void zlabrd_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *nb, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, double *d, double *e,
             armpl_doublecomplex_t *tauq, armpl_doublecomplex_t *taup,
             armpl_doublecomplex_t *x, const armpl_int_t *ldx,
             armpl_doublecomplex_t *y, const armpl_int_t *ldy);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows in the matrix A.

N Input parameter.

N is INTEGER

The number of columns in the matrix A.

NB Input parameter.

NB is INTEGER

The number of leading rows and columns of A to be reduced.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the m by n general matrix to be reduced. On exit, the first NB rows and columns of the matrix are overwritten; the rest of the array is unchanged. If m >= n, elements on and below the diagonal in the first NB columns, with the array TAUQ, represent the unitary matrix Q as a product of elementary reflectors; and elements above the diagonal in the first NB rows, with the array TAUP, represent the unitary matrix P as a product of elementary reflectors. If m < n, elements below the diagonal in the first NB columns, with the array TAUQ, represent the unitary matrix Q as a product of elementary reflectors, and elements on and above the diagonal in the first NB rows, with the array TAUP, represent the unitary matrix P as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

D Output parameter.

D is DOUBLE PRECISION

D is an array, dimension (NB). The diagonal elements of the first NB rows and columns of the reduced matrix. D(i) = A(i,i).

E Output parameter.

E is DOUBLE PRECISION

E is an array, dimension (NB). The off-diagonal elements of the first NB rows and columns of the reduced matrix.

TAUQ Output parameter.

TAUQ is COMPLEX*16

TAUQ is an array, dimension (NB). The scalar factors of the elementary reflectors which represent the unitary matrix Q. See Further Details.

TAUP Output parameter.

TAUP is COMPLEX*16

TAUP is an array, dimension (NB). The scalar factors of the elementary reflectors which represent the unitary matrix P. See Further Details.

X Output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NB). The m-by-nb matrix X required to update the unreduced part of A.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(1, M).

Y Output parameter.

Y is COMPLEX*16

Y is an array, dimension (LDY, NB). The n-by-nb matrix Y required to update the unreduced part of A.

LDY Input parameter.

LDY is INTEGER

The leading dimension of the array Y. LDY >= max(1, N).

Related Information

For this routine in other precisions, please see *clabrd*, *dlabrd* and *slabrd*.

4.17.678 zlacgv

zlacgv conjugates a complex vector of length N.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlacgv(N, X, INCX)
```

C specification:

```
#include "armpl.h"

void zlacgv_(const armpl_int_t *n, armpl_doublecomplex_t *x,
             const armpl_int_t *incx);
```

Parameters

N Input parameter.

N is INTEGER

The length of the vector X. N >= 0.

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension. (1+(N-1)*abs(INCX)) On entry, the vector of length N to be conjugated. On exit, X is overwritten with conjg(X).

INCX Input parameter.

INCX is INTEGER

The spacing between successive elements of X.

Related Information

For this routine in other precisions, please see *clacgv*. It also exists with a native C interface as *LAPACKE_zlacgv*.

4.17.679 zlacn2

`zlacn2` estimates the 1-norm of a square, complex matrix A. Reverse communication is used for evaluating matrix-vector products.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlacn2(N, V, X, EST, KASE, ISAVE)
```

C specification:

```
#include "armpl.h"

void zlacn2_(const armpl_int_t *n, armpl_doublecomplex_t *v,
             armpl_doublecomplex_t *x, double *est, armpl_int_t *kase,
             armpl_int_t *isave);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix. N >= 1.

V Output parameter.

V is COMPLEX*16

V is an array, dimension (N). On the final return, V = A*W, where EST = norm(V)/norm(W) (W is not returned).

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension (N). On an intermediate return, X should be overwritten by A * X, if KASE=1, A^H * X, if KASE=2, where A^H is the conjugate transpose of A, and ZLACN2 must be re-called with all the other parameters unchanged.

EST Input and output parameter.

EST is DOUBLE PRECISION

On entry with KASE = 1 or 2 and ISAVE(1) = 3, EST should be unchanged from the previous call to ZLACN2. On exit, EST is an estimate (a lower bound) for norm(A).

KASE Input and output parameter.

KASE is INTEGER

On the initial call to ZLACN2, KASE should be 0. On an intermediate return, KASE will be 1 or 2, indicating whether X should be overwritten by A * X or A^H * X. On the final return from ZLACN2, KASE will again be 0.

ISAVE Input and output parameter.

ISAVE is INTEGER array, dimension (3)

ISAVE is used to save variables between calls to ZLACN2

Related Information

For this routine in other precisions, please see *clacn2*, *dlacn2* and *slacn2*. It also exists with a native C interface as *LAPACKE_zlacn2*.

4.17.680 zlacon

`zlacon` estimates the 1-norm of a square, complex matrix A. Reverse communication is used for evaluating matrix-vector products.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlacon(N, V, X, EST, KASE)
```

C specification:

```
#include "armpl.h"

void zlacon_(const armpl_int_t *n, armpl_doublecomplex_t *v,
             armpl_doublecomplex_t *x, double *est, armpl_int_t *kase);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix. N >= 1.

V Output parameter.

V is COMPLEX*16

V is an array, dimension (N). On the final return, V = A*W, where EST = norm(V)/norm(W) (W is not returned).

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension (N). On an intermediate return, X should be overwritten by A * X, if KASE=1, A^H * X, if KASE=2, where A^H is the conjugate transpose of A, and ZLACON must be re-called with all the other parameters unchanged.

EST Input and output parameter.

EST is DOUBLE PRECISION

On entry with KASE = 1 or 2 and JUMP = 3, EST should be unchanged from the previous call to ZLACON. On exit, EST is an estimate (a lower bound) for norm(A).

KASE Input and output parameter.

KASE is INTEGER

On the initial call to ZLACON, KASE should be 0. On an intermediate return, KASE will be 1 or 2, indicating whether X should be overwritten by A * X or A^H * X. On the final return from ZLACON, KASE will again be 0.

Related Information

For this routine in other precisions, please see *clacon*, *dlacon* and *slacon*.

4.17.681 zlacp2

`zlacp2` copies all or part of a real two-dimensional matrix A to a complex matrix B.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlacp2(UPLO, M, N, A, LDA, B, LDB)
```

C specification:

```
#include "armpl.h"

void zlacp2_(const char *uplo, const armpl_int_t *m, const armpl_int_t *n,
             const double *a, const armpl_int_t *lda,
             armpl_doublecomplex_t *b, const armpl_int_t *ldb, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies the part of the matrix A to be copied to B. = 'U': Upper triangular part = 'L': Lower triangular part Otherwise: All of the matrix A

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, N). The m by n matrix A. If UPLO = 'U', only the upper trapezium is accessed; if UPLO = 'L', only the lower trapezium is accessed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). On exit, B = A in the locations specified by UPLO.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M).

Related Information

For this routine in other precisions, please see *clacp2*. It also exists with a native C interface as *LAPACKE_zlacp2*.

4.17.682 zlacpy

zlacpy copies all or part of a two-dimensional matrix A to another matrix B.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlacpy(UPLO, M, N, A, LDA, B, LDB)
```

C specification:

```
#include "armpl.h"

void zlacpy_(const char *uplo, const armpl_int_t *m, const armpl_int_t *n,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_doublecomplex_t *b, const armpl_int_t *ldb, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies the part of the matrix A to be copied to B. = 'U': Upper triangular part = 'L': Lower triangular part Otherwise: All of the matrix A

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The m by n matrix A. If UPLO = 'U', only the upper trapezium is accessed; if UPLO = 'L', only the lower trapezium is accessed.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). On exit, B = A in the locations specified by UPLO.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M).

Related Information

For this routine in other precisions, please see *clacpy*, *dlacpy* and *slacpy*. It also exists with a native C interface as *LAPACKE_zlacpy*.

4.17.683 zlacrm

zlacrm performs a very simple matrix-matrix multiplication:

```
C := A * B,
```

where A is M by N and complex; B is N by N and real; C is M by N and complex.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlacrm(M, N, A, LDA, B, LDB, C, LDC, RWORK)
```

C specification:

```
#include "armpl.h"

void zlacrm_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const double *b, const armpl_int_t *ldb,
             armpl_doublecomplex_t *c, const armpl_int_t *ldc,
             double *rwork);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A and of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns and rows of the matrix B and the number of columns of the matrix C. N >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, A contains the M by N matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >=max(1, M).

B Input parameter.

B is DOUBLE PRECISION

B is an array, dimension (LDB, N). On entry, B contains the N by N matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >=max(1, N).

C Output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). On exit, C contains the M by N matrix C.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >=max(1, N).

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (2*M*N) .

Related Information

For this routine in other precisions, please see *clacrm*. It also exists with a native C interface as *LAPACKE_zlacrm*.

4.17.684 zlacrt

zlacrt performs the operation

```
(  c  s )( x )  ==> ( x )
( -s  c )( y )      ( y )
```

where c and s are complex and the vectors x and y are complex.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlacrt(N, CX, INCX, CY, INCY, C, S)
```

C specification:

```
#include "armpl.h"

void zlacrt_(const armpl_int_t *n, armpl_doublecomplex_t *cx,
             const armpl_int_t *incx, armpl_doublecomplex_t *cy,
             const armpl_int_t *incy, const armpl_doublecomplex_t *c,
             const armpl_doublecomplex_t *s);
```

Parameters

N Input parameter.

N is INTEGER

The number of elements in the vectors CX and CY.

CX Input and output parameter.

CX is COMPLEX*16

CX is an array, dimension (N). On input, the vector x. On output, CX is overwritten with c*x + s*y.

INCX Input parameter.

INCX is INTEGER

The increment between successive values of CX. INCX <> 0.

CY Input and output parameter.

CY is COMPLEX*16

CY is an array, dimension (N). On input, the vector y. On output, CY is overwritten with -s*x + c*y.

INCY Input parameter.

INCY is INTEGER

The increment between successive values of CY. INCY <> 0.

C Input parameter.

C is COMPLEX*16

S Input parameter.

S is COMPLEX*16

C and S define the matrix [C S]. [-S C]

Related Information

For this routine in other precisions, please see *clacrt*.

4.17.685 zladiv

zladiv := X / Y, where X and Y are complex. The computation of X / Y will not overflow on an intermediary step unless the results overflows.

Syntax

Fortran specification:

```
use armpl_library

complex*16 function zladiv(X, Y)
```

C specification:

```
#include "armpl.h"

DOUBLECOMPLEX_RET_VALUE zladiv_(DOUBLECOMPLEX_RET_PARAM const armpl_doublecomplex_
↪t *x,
                                const armpl_doublecomplex_t *y);
```

Parameters

X Input parameter.

X is COMPLEX*16

Y Input parameter.

Y is COMPLEX*16

The complex scalars X and Y.

Related Information

For this routine in other precisions, please see *cladiv*, *dladiv* and *sladiv*.

4.17.686 zlaed0

Using the divide and conquer method, zlaed0 computes all eigenvalues of a symmetric tridiagonal matrix which is one diagonal block of those from reducing a dense or band Hermitian matrix and corresponding eigenvectors of the dense or band matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlaed0(QSIZ, N, D, E, Q, LDQ, QSTORE, LDQS, RWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zlaed0_(const armpl_int_t *qsiz, const armpl_int_t *n, double *d,
             double *e, armpl_doublecomplex_t *q, const armpl_int_t *ldq,
             armpl_doublecomplex_t *qstore, const armpl_int_t *ldqs,
             double *rwork, armpl_int_t *iwork, armpl_int_t *info);
```

Parameters

QSIZ Input parameter.

QSIZ is INTEGER

The dimension of the unitary matrix used to reduce the full matrix to tridiagonal form. QSIZ >= N if ICOMPQ = 1.

N Input parameter.

N is INTEGER

The dimension of the symmetric tridiagonal matrix. N >= 0.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry, the diagonal elements of the tridiagonal matrix. On exit, the eigenvalues in ascending order.

E Input and output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N-1). On entry, the off-diagonal elements of the tridiagonal matrix. On exit, E has been destroyed.

Q Input and output parameter.

Q is COMPLEX*16

Q is an array, dimension (LDQ, N). On entry, Q must contain an QSIZ x N matrix whose columns unitarily orthonormal. It is a part of the unitary matrix that reduces the full dense Hermitian matrix to a (reducible) symmetric tridiagonal matrix.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, N).

IWORK Output parameter.

IWORK is INTEGER array,

the dimension of IWORK must be at least 6 + 6* N + 5*N*lg N (lg(N) = smallest integer k such that 2^k >= N)

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array,. dimension (1 + 3* N + 2*N*lg N + 3*N**2) (lg(N) = smallest integer k such that 2^k >= N)

QSTORE Output parameter.

QSTORE is COMPLEX*16

QSTORE is an array, dimension (LDQS, N). Used to store parts of the eigenvector matrix when the updating matrix multiplies take place.

LDQS Input parameter.

LDQS is INTEGER

The leading dimension of the array QSTORE. LDQS >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: The algorithm failed to compute an eigenvalue while working on the submatrix lying in rows and columns INFO/(N+1) through mod(INFO,N+1).

Related Information

For this routine in other precisions, please see *claed0*, *dlaed0* and *slaed0*.

4.17.687 zlaed7

zlaed7 computes the updated eigensystem of a diagonal matrix after modification by a rank-one symmetric matrix. This routine is used only for the eigenproblem which requires all eigenvalues and optionally eigenvectors of a dense or banded Hermitian matrix that has been reduced to tridiagonal form.

```
T = Q(in) ( D(in) + RHO * Z*Z**H ) Q**H(in) = Q(out) * D(out) * Q**H(out)

where Z = Q**Hu, u is a vector of length N with ones in the
CUTPNT and CUTPNT + 1 th elements and zeros elsewhere.

 The eigenvectors of the original matrix are stored in Q, and the
 eigenvalues are in D.  The algorithm consists of three stages:

    The first stage consists of deflating the size of the problem
    when there are multiple eigenvalues or if there is a zero in
    the Z vector.  For each such occurrence the dimension of the
    secular equation problem is reduced by one.  This stage is
    performed by the routine DLAED2.

    The second stage consists of calculating the updated
    eigenvalues. This is done by finding the roots of the secular
    equation via the routine DLAED4 (as called by SLAED3).
    This routine also calculates the eigenvectors of the current
    problem.

    The final stage consists of computing the updated eigenvectors
    directly using the updated eigenvalues.  The eigenvectors for
    the current problem are multiplied with the eigenvectors from
    the overall problem.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zlaed7(N, CUTPNT, QSIZ, TLVLS, CURLVL, CURPBM, D, Q, LDQ, RHO,
                  INDXQ, QSTORE, QPTR, PRMPTR, PERM, GIVPTR, GIVCOL, GIVNUM,
                  WORK, RWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zlaed7_(const armpl_int_t *n, const armpl_int_t *cutpnt,
             const armpl_int_t *qsiz, const armpl_int_t *tlvls,
             const armpl_int_t *curlvl, const armpl_int_t *curpbm, double *d,
             armpl_doublecomplex_t *q, const armpl_int_t *ldq,
             const double *rho, armpl_int_t *indxq, double *qstore,
             armpl_int_t *qptr, const armpl_int_t *prmptr,
             const armpl_int_t *perm, const armpl_int_t *givptr,
             const armpl_int_t *givcol, const double *givnum,
             armpl_doublecomplex_t *work, double *rwork, armpl_int_t *iwork,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The dimension of the symmetric tridiagonal matrix. N >= 0.

CUTPNT Input parameter.

CUTPNT is INTEGER

Contains the location of the last eigenvalue in the leading sub-matrix. min(1, N) <= CUTPNT <= N.

QSIZ Input parameter.

QSIZ is INTEGER

The dimension of the unitary matrix used to reduce the full matrix to tridiagonal form. QSIZ >= N.

TLVLS Input parameter.

TLVLS is INTEGER

The total number of merging levels in the overall divide and conquer tree.

CURLVL Input parameter.

CURLVL is INTEGER

The current level in the overall merge routine, 0 <= curlvl <= tlvls.

CURPBM Input parameter.

CURPBM is INTEGER

The current problem in the current level in the overall merge routine (counting from upper left to lower right).

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry, the eigenvalues of the rank-1-perturbed matrix. On exit, the eigenvalues of the repaired matrix.

Q Input and output parameter.

Q is COMPLEX*16

Q is an array, dimension (LDQ, N). On entry, the eigenvectors of the rank-1-perturbed matrix. On exit, the eigenvectors of the repaired tridiagonal matrix.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, N).

RHO Input parameter.

RHO is DOUBLE PRECISION

Contains the subdiagonal element used to create the rank-1 modification.

INDXQ Output parameter.

INDXQ is INTEGER array, dimension (N)

This contains the permutation which will reintegrate the subproblem just solved back into sorted order, ie. D(INDXQ(I = 1, N)) will be in ascending order.

IWORK Output parameter.

IWORK is INTEGER array, dimension (4*N)

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array,. dimension (3*N+2*QSIZ*N)

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (QSIZ*N) .

QSTORE Input and output parameter.

QSTORE is DOUBLE PRECISION

QSTORE is an array, dimension (N**2+1). Stores eigenvectors of submatrices encountered during divide and conquer, packed together. QPTR points to beginning of the submatrices.

QPTR Input and output parameter.

QPTR is INTEGER array, dimension (N+2)

List of indices pointing to beginning of submatrices stored in QSTORE. The submatrices are numbered starting at the bottom left of the divide and conquer tree, from left to right and bottom to top.

PRMPTR Input parameter.

PRMPTR is INTEGER array, dimension (N lg N)

Contains a list of pointers which indicate where in PERM a level's permutation is stored. PRMPTR(i+1) - PRMPTR(i) indicates the size of the permutation and also the size of the full, non-deflated problem.

PERM Input parameter.

PERM is INTEGER array, dimension (N lg N)

Contains the permutations (from deflation and sorting) to be applied to each eigenblock.

GIVPTR Input parameter.

GIVPTR is INTEGER array, dimension (N lg N)

Contains a list of pointers which indicate where in GIVCOL a level's Givens rotations are stored. GIVPTR(i+1) - GIVPTR(i) indicates the number of Givens rotations.

GIVCOL Input parameter.

GIVCOL is INTEGER array, dimension (2, N lg N)

Each pair of numbers indicates a pair of columns to take place in a Givens rotation.

GIVNUM Input parameter.

GIVNUM is DOUBLE PRECISION

GIVNUM is an array, dimension (2, N lg N). Each number indicates the S value to be used in the corresponding Givens rotation.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: if INFO = 1, an eigenvalue did not converge

Related Information

For this routine in other precisions, please see *claed7*, *dlaed7* and *slaed7*.

4.17.688 zlaed8

zlaed8 merges the two sets of eigenvalues together into a single sorted set. Then it tries to deflate the size of the problem. There are two ways in which deflation can occur: when two or more eigenvalues are close together or if there is a tiny element in the Z vector. For each such occurrence the order of the related secular equation problem is reduced by one.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlaed8(K, N, QSIZ, Q, LDQ, D, RHO, CUTPNT, Z, DLAMDA, Q2, LDQ2, W,
                  INDXP, INDX, INDXQ, PERM, GIVPTR, GIVCOL, GIVNUM, INFO)
```

C specification:

```
#include "armpl.h"

void zlaed8_(armpl_int_t *k, const armpl_int_t *n, const armpl_int_t *qsiz,
             armpl_doublecomplex_t *q, const armpl_int_t *ldq, double *d,
             double *rho, const armpl_int_t *cutpnt, const double *z,
             double *dlamda, armpl_doublecomplex_t *q2,
             const armpl_int_t *ldq2, double *w, armpl_int_t *indxp,
             armpl_int_t *indx, const armpl_int_t *indxq, armpl_int_t *perm,
             armpl_int_t *givptr, armpl_int_t *givcol, double *givnum,
             armpl_int_t *info);
```

Parameters

K Output parameter.

K is INTEGER

Contains the number of non-deflated eigenvalues. This is the order of the related secular equation.

N Input parameter.

N is INTEGER

The dimension of the symmetric tridiagonal matrix. N >= 0.

QSIZ Input parameter.

QSIZ is INTEGER

The dimension of the unitary matrix used to reduce the dense or band matrix to tridiagonal form. QSIZ >= N if ICOMPQ = 1.

Q Input and output parameter.

Q is COMPLEX*16

Q is an array, dimension (LDQ, N). On entry, Q contains the eigenvectors of the partially solved system which has been previously updated in matrix multiplies with other partially solved eigensystems. On exit, Q contains the trailing (N-K) updated eigenvectors (those which were deflated) in its last N-K columns.

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= max(1, N).

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry, D contains the eigenvalues of the two submatrices to be combined. On exit, D contains the trailing (N-K) updated eigenvalues (those which were deflated) sorted into increasing order.

RHO Input and output parameter.

RHO is DOUBLE PRECISION

Contains the off diagonal element associated with the rank-1 cut which originally split the two submatrices which are now being recombined. RHO is modified during the computation to the value required by DLAED3.

CUTPNT Input parameter.

CUTPNT is INTEGER

Contains the location of the last eigenvalue in the leading sub-matrix. MIN(1, N) <= CUTPNT <= N.

Z Input parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (N). On input this vector contains the updating vector (the last row of the first sub-eigenvector matrix and the first row of the second sub-eigenvector matrix). The contents of Z are destroyed during the updating process.

DLAMDA Output parameter.

DLAMDA is DOUBLE PRECISION

DLAMDA is an array, dimension (N). Contains a copy of the first K eigenvalues which will be used by DLAED3 to form the secular equation.

Q2 Output parameter.

Q2 is COMPLEX*16

Q2 is an array, dimension (LDQ2, N). If ICOMPQ = 0, Q2 is not referenced. Otherwise, Contains a copy of the first K eigenvectors which will be used by DLAED7 in a matrix multiply (DGEMM) to update the new eigenvectors.

LDQ2 Input parameter.

LDQ2 is INTEGER

The leading dimension of the array Q2. LDQ2 >= max(1, N).

W Output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). This will hold the first k values of the final deflation-altered z-vector and will be passed to DLAED3.

INDXP Output parameter.

INDXP is INTEGER array, dimension (N)

This will contain the permutation used to place deflated values of D at the end of the array. On output INDXP(1:K) points to the nondeflated D-values and INDXP(K+1:N) points to the deflated eigenvalues.

INDX Output parameter.

INDX is INTEGER array, dimension (N)

This will contain the permutation used to sort the contents of D into ascending order.

INDXQ Input parameter.

INDXQ is INTEGER array, dimension (N)

This contains the permutation which separately sorts the two sub-problems in D into ascending order. Note that elements in the second half of this permutation must first have CUTPNT added to their values in order to be accurate.

PERM Output parameter.

PERM is INTEGER array, dimension (N)

Contains the permutations (from deflation and sorting) to be applied to each eigenblock.

GIVPTR Output parameter.

GIVPTR is INTEGER

Contains the number of Givens rotations which took place in this subproblem.

GIVCOL Output parameter.

GIVCOL is INTEGER array, dimension (2, N)

Each pair of numbers indicates a pair of columns to take place in a Givens rotation.

GIVNUM Output parameter.

GIVNUM is DOUBLE PRECISION

GIVNUM is an array, dimension (2, N). Each number indicates the S value to be used in the corresponding Givens rotation.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *claed8*, *dlaed8* and *slaed8*.

4.17.689 zlaein

zlaein uses inverse iteration to find a right or left eigenvector corresponding to the eigenvalue W of a complex upper Hessenberg matrix H.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlaein(RIGHTV, NOINIT, N, H, LDH, W, V, B, LDB, RWORK, EPS3,
                  SMLNUM, INFO)
```

C specification:

```
#include "armpl.h"

void zlaein_(const armpl_int_t *rightv, const armpl_int_t *noinit,
             const armpl_int_t *n, const armpl_doublecomplex_t *h,
             const armpl_int_t *ldh, const armpl_doublecomplex_t *w,
             armpl_doublecomplex_t *v, armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, double *rwork, const double *eps3,
             const double *smlnum, armpl_int_t *info);
```

Parameters

RIGHTV Input parameter.

RIGHTV is LOGICAL

= .TRUE. : compute right eigenvector; = .FALSE.: compute left eigenvector.

NOINIT Input parameter.

NOINIT is LOGICAL

= .TRUE. : no initial vector supplied in V = .FALSE.: initial vector supplied in V.

N Input parameter.

N is INTEGER

The order of the matrix H. N >= 0.

H Input parameter.

H is COMPLEX*16

H is an array, dimension (LDH, N). The upper Hessenberg matrix H.

LDH Input parameter.

LDH is INTEGER

The leading dimension of the array H. LDH >= max(1, N).

W Input parameter.

W is COMPLEX*16

The eigenvalue of H whose corresponding right or left eigenvector is to be computed.

V Input and output parameter.

V is COMPLEX*16

V is an array, dimension (N). On entry, if NOINIT = .FALSE., V must contain a starting vector for inverse iteration; otherwise V need not be set. On exit, V contains the computed eigenvector, normalized so that the component of largest magnitude has magnitude 1; here the magnitude of a complex number (x,y) is taken to be |x| + |y|.

B Output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N) .

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (N) .

EPS3 Input parameter.

EPS3 is DOUBLE PRECISION

A small machine-dependent value which is used to perturb close eigenvalues, and to replace zero pivots.

SMLNUM Input parameter.

SMLNUM is DOUBLE PRECISION

A machine-dependent value close to the underflow threshold.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit = 1: inverse iteration did not converge; V is set to the last iterate.

Related Information

For this routine in other precisions, please see *claein*, *dlaein* and *slaein*.

4.17.690 zlaesy

`zlaesy` computes the eigendecomposition of a 2-by-2 symmetric matrix

```
( ( A, B );( B, C ) )
```

provided the norm of the matrix of eigenvectors is larger than some threshold value.

RT1 is the eigenvalue of larger absolute value, and RT2 of smaller absolute value. If the eigenvectors are computed, then on return (CS1, SN1) is the unit eigenvector for RT1, hence

[CS1 SN1] . [A B] . [CS1 -SN1] = [RT1 0] [-SN1 CS1] [B C] [SN1 CS1] [0 RT2]

Syntax

Fortran specification:

```
use armpl_library

subroutine zlaesy(A, B, C, RT1, RT2, EVSCAL, CS1, SN1)
```

C specification:

```
#include "armpl.h"

void zlaesy_(const armpl_doublecomplex_t *a, const armpl_doublecomplex_t *b,
             const armpl_doublecomplex_t *c, armpl_doublecomplex_t *rt1,
             armpl_doublecomplex_t *rt2, armpl_doublecomplex_t *evscal,
             armpl_doublecomplex_t *cs1, armpl_doublecomplex_t *sn1);
```

Parameters

A Input parameter.

`A` is `COMPLEX*16`

The (1, 1) element of input matrix.

B Input parameter.

`B` is `COMPLEX*16`

The (1, 2) element of input matrix. The (2, 1) element is also given by `B`, since the 2-by-2 matrix is symmetric.

C Input parameter.

`C` is `COMPLEX*16`

The (2, 2) element of input matrix.

RT1 Output parameter.

`RT1` is `COMPLEX*16`

The eigenvalue of larger modulus.

RT2 Output parameter.

RT2 is COMPLEX*16

The eigenvalue of smaller modulus.

EVSCAL Output parameter.

EVSCAL is COMPLEX*16

The complex value by which the eigenvector matrix was scaled to make it orthonormal. If EVSCAL is zero, the eigenvectors were not computed. This means one of two things: the 2-by-2 matrix could not be diagonalized, or the norm of the matrix of eigenvectors before scaling was larger than the threshold value THRESH (set below).

CS1 Output parameter.

CS1 is COMPLEX*16

SN1 Output parameter.

SN1 is COMPLEX*16

If EVSCAL .NE. 0, (CS1, SN1) is the unit right eigenvector for RT1.

Related Information

For this routine in other precisions, please see *claesy*.

4.17.691 zlaev2

zlaev2 computes the eigendecomposition of a 2-by-2 Hermitian matrix

```
[  A         B  ]
[  CONJG(B)  C  ].
```

On return, RT1 is the eigenvalue of larger absolute value, RT2 is the eigenvalue of smaller absolute value, and (CS1,SN1) is the unit right eigenvector for RT1, giving the decomposition

[CS1 CONJG(SN1)] [A B] [CS1 -CONJG(SN1)] = [RT1 0] [-SN1 CS1] [CONJG(B) C] [SN1 CS1] [0 RT2].

Syntax

Fortran specification:

```
use armpl_library

subroutine zlaev2(A, B, C, RT1, RT2, CS1, SN1)
```

C specification:

```
#include "armpl.h"

void zlaev2_(const armpl_doublecomplex_t *a, const armpl_doublecomplex_t *b,
             const armpl_doublecomplex_t *c, double *rt1, double *rt2,
             double *cs1, armpl_doublecomplex_t *sn1);
```

Parameters

A Input parameter.

A is COMPLEX*16

The (1,1) element of the 2-by-2 matrix.

B Input parameter.

B is COMPLEX*16

The (1,2) element and the conjugate of the (2,1) element of the 2-by-2 matrix.

C Input parameter.

C is COMPLEX*16

The (2,2) element of the 2-by-2 matrix.

RT1 Output parameter.

RT1 is DOUBLE PRECISION

The eigenvalue of larger absolute value.

RT2 Output parameter.

RT2 is DOUBLE PRECISION

The eigenvalue of smaller absolute value.

CS1 Output parameter.

CS1 is DOUBLE PRECISION

SN1 Output parameter.

SN1 is COMPLEX*16

The vector (CS1, SN1) is a unit right eigenvector for RT1.

Related Information

For this routine in other precisions, please see *claev2*, *dlaev2* and *slaev2*.

4.17.692 zlag2c

zlag2c converts a COMPLEX*16 matrix, SA, to a COMPLEX matrix, A.

RMAX is the overflow for the SINGLE PRECISION arithmetic zlag2c checks that all the entries of A are between -RMAX and RMAX. If not the conversion is aborted and a flag is raised.

This is an auxiliary routine so there is no argument checking.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlag2c(M, N, A, LDA, SA, LDSA, INFO)
```

C specification:

```
#include "armpl.h"

void zlag2c_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_singlecomplex_t *sa, const armpl_int_t *ldsa,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of lines of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the M-by-N coefficient matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

SA Output parameter.

SA is COMPLEX

SA is an array, dimension (LDSA, N). On exit, if INFO=0, the M-by-N coefficient matrix SA; if INFO>0, the content of SA is unspecified.

LDSA Input parameter.

LDSA is INTEGER

The leading dimension of the array SA. LDSA >= max(1, M).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. = 1: an entry of the matrix A is greater than the SINGLE PRECISION overflow threshold, in this case, the content of SA in exit is unspecified.

Related Information

It also exists with a native C interface as *LAPACKE_zlag2c*.

4.17.693 zlags2

zlags2 computes 2-by-2 unitary matrices U, V and Q, such that if (UPPER) then

```
U**H *A*Q = U**H *( A1 A2 )*Q = ( x  0  )
                  ( 0  A3 )     ( x  x  )
```

and

```
V**H*B*Q = V**H *( B1  B2 )*Q = ( x   0  )
                ( 0   B3 )      ( x   x  )
```

or if (.NOT.UPPER) then

```
U**H *A*Q = U**H *( A1  0  )*Q = ( x   x  )
                 ( A2  A3 )      ( 0   x  )
```

and

```
V**H *B*Q = V**H *( B1  0  )*Q = ( x   x  )
                 ( B2  B3 )      ( 0   x  )
```

where

```
 U = (    CSU     SNU ), V = (  CSV     SNV ),
     ( -SNU**H   CSU )       ( -SNV**H CSV )

Q = (    CSQ     SNQ )
     ( -SNQ**H   CSQ )
```

The rows of the transformed A and B are parallel. Moreover, if the input 2-by-2 matrix A is not zero, then the transformed (1,1) entry of A is not zero. If the input matrices A and B are both not zero, then the transformed (2,2) element of B is not zero, except when the first rows of input A and B are parallel and the second rows are zero.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlags2(UPPER, A1, A2, A3, B1, B2, B3, CSU, SNU, CSV, SNV, CSQ,
                  SNQ)
```

C specification:

```
#include "armpl.h"

void zlags2_(const armpl_int_t *upper, const double *a1,
             const armpl_doublecomplex_t *a2, const double *a3,
             const double *b1, const armpl_doublecomplex_t *b2,
             const double *b3, double *csu, armpl_doublecomplex_t *snu,
             double *csv, armpl_doublecomplex_t *snv, double *csq,
             armpl_doublecomplex_t *snq);
```

Parameters

UPPER Input parameter.

UPPER is LOGICAL

= .TRUE.: the input matrices A and B are upper triangular. = .FALSE.: the input matrices A and B are lower triangular.

A1 Input parameter.

A1 is DOUBLE PRECISION

A2 Input parameter.

A2 is COMPLEX*16

A3 Input parameter.

A3 is DOUBLE PRECISION

On entry, A1, A2 and A3 are elements of the input 2-by-2 upper (lower) triangular matrix A.

B1 Input parameter.

B1 is DOUBLE PRECISION

B2 Input parameter.

B2 is COMPLEX*16

B3 Input parameter.

B3 is DOUBLE PRECISION

On entry, B1, B2 and B3 are elements of the input 2-by-2 upper (lower) triangular matrix B.

CSU Output parameter.

CSU is DOUBLE PRECISION

SNU Output parameter.

SNU is COMPLEX*16

The desired unitary matrix U.

CSV Output parameter.

CSV is DOUBLE PRECISION

SNV Output parameter.

SNV is COMPLEX*16

The desired unitary matrix V.

CSQ Output parameter.

CSQ is DOUBLE PRECISION

SNQ Output parameter.

SNQ is COMPLEX*16

The desired unitary matrix Q.

Related Information

For this routine in other precisions, please see *clags2*, *dlags2* and *slags2*.

4.17.694 zlagtm

zlagtm performs a matrix-vector product of the form

```
B := alpha * A * X + beta * B
```

where A is a tridiagonal matrix of order N, B and X are N by NRHS matrices, and alpha and beta are real scalars, each of which may be 0., 1., or -1.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlagtm(TRANS, N, NRHS, ALPHA, DL, D, DU, X, LDX, BETA, B, LDB)
```

C specification:

```
#include "armpl.h"

void zlagtm_(const char *trans, const armpl_int_t *n, const armpl_int_t *nrhs,
             const double *alpha, const armpl_doublecomplex_t *dl,
             const armpl_doublecomplex_t *d, const armpl_doublecomplex_t *du,
             const armpl_doublecomplex_t *x, const armpl_int_t *ldx,
             const double *beta, armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the operation applied to A. = 'N': No transpose, B := alpha * A * X + beta * B = 'T': Transpose, B := alpha * A^T * X + beta * B = 'C': Conjugate transpose, B := alpha * A^H * X + beta * B

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrices X and B.

ALPHA Input parameter.

ALPHA is DOUBLE PRECISION

The scalar alpha. ALPHA must be 0., 1., or -1.; otherwise, it is assumed to be 0.

DL Input parameter.

DL is COMPLEX*16

DL is an array, dimension (N-1). The (n-1) sub-diagonal elements of T.

D Input parameter.

D is COMPLEX*16

D is an array, dimension (N). The diagonal elements of T.

DU Input parameter.

DU is COMPLEX*16

DU is an array, dimension (N-1). The (n-1) super-diagonal elements of T.

X Input parameter.

X is COMPLEX*16

X is an array, dimension (LDX, NRHS). The N by NRHS matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X. LDX >= max(N,1).

BETA Input parameter.

BETA is DOUBLE PRECISION

The scalar beta. BETA must be 0., 1., or -1.; otherwise, it is assumed to be 1.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the N by NRHS matrix B. On exit, B is overwritten by the matrix expression B := alpha * A * X + beta * B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(N,1).

Related Information

For this routine in other precisions, please see *clagtm*, *dlagtm* and *slagtm*.

4.17.695 zlahef

zlahef computes a partial factorization of a complex Hermitian matrix A using the Bunch-Kaufman diagonal pivoting method. The partial factorization has the form:

A = (I U12) (A11 0) (I 0) if UPLO = 'U', or:

```
( 0  U22 ) (  0   D  ) ( U12**H U22**H )
```

A = (L11 0) (D 0) ($L11^H$ $L21^H$) if UPLO = 'L'

```
( L21  I ) (  0  A22 ) (  0      I     )
```

where the order of D is at most NB. The actual order is returned in the argument KB, and is either NB or NB-1, or N if N <= NB. Note that U^H denotes the conjugate transpose of U.

zlahef is an auxiliary routine called by ZHETRF. It uses blocked code (calling Level 3 BLAS) to update the submatrix A11 (if UPLO = 'U') or A22 (if UPLO = 'L').

Syntax

Fortran specification:

```
use armpl_library

subroutine zlahef(UPLO, N, NB, KB, A, LDA, IPIV, W, LDW, INFO)
```

C specification:

```
#include "armpl.h"

void zlahef_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nb,
             armpl_int_t *kb, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *ipiv,
```

(continues on next page)

(continued from previous page)

```
            armpl_doublecomplex_t *w, const armpl_int_t *ldw,
            armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the Hermitian matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NB Input parameter.

NB is INTEGER

The maximum number of columns of the matrix A that should be factored. NB should be at least 2 to allow for 2-by-2 pivot blocks.

KB Output parameter.

KB is INTEGER

The number of columns of A that were actually factored. KB is either NB-1 or NB, or N if N <= NB.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, A contains details of the partial factorization.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D.

If UPLO = 'U': Only the last KB elements of IPIV are set.

If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block.

If UPLO = 'L': Only the first KB elements of IPIV are set.

If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

W Output parameter.

W is COMPLEX*16

W is an array, dimension (LDW, NB) .

LDW Input parameter.

LDW is INTEGER

The leading dimension of the array W. LDW >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit > 0: if INFO = k, D(k,k) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular.

Related Information

For this routine in other precisions, please see *clahef*.

4.17.696 zlahef_aa

DLAHEF_AA factorizes a panel of a complex hermitian matrix A using the Aasen's algorithm. The panel consists of a set of NB rows of A when UPLO is U, or a set of NB columns when UPLO is L.

In order to factorize the panel, the Aasen's algorithm requires the last row, or column, of the previous panel. The first row, or column, of A is set to be the first row, or column, of an identity matrix, which is used to factorize the first panel.

The resulting J-th row of U, or J-th column of L, is stored in the (J-1)-th row, or column, of A (without the unit diagonals), while the diagonal and subdiagonal of A are overwritten by those of T.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlahef_aa(UPLO, J1, M, NB, A, LDA, IPIV, H, LDH, WORK)
```

C specification:

```
#include "armpl.h"

void zlahef_aa_(const char *uplo, const armpl_int_t *j1, const armpl_int_t *m,
                const armpl_int_t *nb, armpl_doublecomplex_t *a,
                const armpl_int_t *lda, armpl_int_t *ipiv,
                armpl_doublecomplex_t *h, const armpl_int_t *ldh,
                armpl_doublecomplex_t *work, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

J1 Input parameter.

J1 is INTEGER

The location of the first row, or column, of the panel within the submatrix of A, passed to this routine, e.g., when called by ZHETRF_AA, for the first panel, J1 is 1, while for the remaining panels, J1 is 2.

M Input parameter.

M is INTEGER

The dimension of the submatrix. M >= 0.

NB Input parameter.

NB is INTEGER

The dimension of the panel to be facotorized.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, M) for. the first panel, while dimension (LDA,M+1) for the remaining panels.

On entry, A contains the last row, or column, of the previous panel, and the trailing submatrix of A to be factorized, except for the first panel, only the panel is passed.

On exit, the leading panel is factorized.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1,N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the row and column interchanges, the row and column k were interchanged with the row and column IPIV(k).

H Input and output parameter.

H is COMPLEX*16 workspace, dimension (LDH, NB).

LDH Input parameter.

LDH is INTEGER

The leading dimension of the workspace H. LDH >= max(1, M).

WORK Output parameter.

WORK is COMPLEX*16 workspace, dimension (M).

Related Information

For this routine in other precisions, please see *clahef_aa*.

4.17.697 zlahef_rook

ZLAHEF_ROOK computes a partial factorization of a complex Hermitian matrix A using the bounded Bunch-Kaufman ("rook") diagonal pivoting method. The partial factorization has the form:

A = (I U12) (A11 0) (I 0) if UPLO = 'U', or:

```
( 0  U22 ) (  0   D  ) ( U12**H U22**H )
```

A = (L11 0) (D 0) ($L11^H$ $L21^H$) if UPLO = 'L'

```
( L21  I ) (  0  A22 ) (  0      I     )
```

where the order of D is at most NB. The actual order is returned in the argument KB, and is either NB or NB-1, or N if N <= NB. Note that U^H denotes the conjugate transpose of U.

ZLAHEF_ROOK is an auxiliary routine called by ZHETRF_ROOK. It uses blocked code (calling Level 3 BLAS) to update the submatrix A11 (if UPLO = 'U') or A22 (if UPLO = 'L').

Syntax

Fortran specification:

```
use armpl_library

subroutine zlahef_rook(UPLO, N, NB, KB, A, LDA, IPIV, W, LDW, INFO)
```

C specification:

```
#include "armpl.h"

void zlahef_rook_(const char *uplo, const armpl_int_t *n,
                  const armpl_int_t *nb, armpl_int_t *kb,
                  armpl_doublecomplex_t *a, const armpl_int_t *lda,
                  armpl_int_t *ipiv, armpl_doublecomplex_t *w,
                  const armpl_int_t *ldw, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the Hermitian matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NB Input parameter.

NB is INTEGER

The maximum number of columns of the matrix A that should be factored. NB should be at least 2 to allow for 2-by-2 pivot blocks.

KB Output parameter.

KB is INTEGER

The number of columns of A that were actually factored. KB is either NB-1 or NB, or N if N <= NB.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, A contains details of the partial factorization.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D.

If UPLO = 'U': Only the last KB elements of IPIV are set.

If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k-1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k-1 and -IPIV(k-1) were inerchaged, D(k-1:k,k-1:k) is a 2-by-2 diagonal block.

If UPLO = 'L': Only the first KB elements of IPIV are set.

If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k+1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k+1 and -IPIV(k+1) were inerchaged, D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

W Output parameter.

W is COMPLEX*16

W is an array, dimension (LDW, NB) .

LDW Input parameter.

LDW is INTEGER

The leading dimension of the array W. LDW >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit > 0: if INFO = k, D(k,k) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular.

Related Information

For this routine in other precisions, please see *clahef_rook*.

4.17.698 zlahqr

```
ZLAHQR is an auxiliary routine called by CHSEQR to update the
   eigenvalues and Schur decomposition already computed by CHSEQR, by
   dealing with the Hessenberg submatrix in rows and columns ILO to
   IHI.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zlahqr(WANTT, WANTZ, N, ILO, IHI, H, LDH, W, ILOZ, IHIZ, Z, LDZ,
                  INFO)
```

C specification:

```
#include "armpl.h"

void zlahqr_(const armpl_int_t *wantt, const armpl_int_t *wantz,
             const armpl_int_t *n, const armpl_int_t *ilo,
             const armpl_int_t *ihi, armpl_doublecomplex_t *h,
             const armpl_int_t *ldh, armpl_doublecomplex_t *w,
             const armpl_int_t *iloz, const armpl_int_t *ihiz,
             armpl_doublecomplex_t *z, const armpl_int_t *ldz,
             armpl_int_t *info);
```

Parameters

WANTT Input parameter.

WANTT is LOGICAL

= .TRUE. : the full Schur form T is required; = .FALSE.: only eigenvalues are required.

WANTZ Input parameter.

WANTZ is LOGICAL

= .TRUE. : the matrix of Schur vectors Z is required; = .FALSE.: Schur vectors are not required.

N Input parameter.

N is INTEGER

The order of the matrix H. N >= 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

It is assumed that H is already upper triangular in rows and columns IHI+1:N, and that H(ILO,ILO-1) = 0 (unless ILO = 1). ZLAHQR works primarily with the Hessenberg submatrix in rows and columns ILO to IHI, but applies transformations to all of H if WANTT is .TRUE.. 1 <= ILO <= max(1, IHI); IHI <= N.

H Input and output parameter.

H is COMPLEX*16

H is an array, dimension (LDH, N). On entry, the upper Hessenberg matrix H. On exit, if INFO is zero and if WANTT is .TRUE., then H is upper triangular in rows and columns ILO:IHI. If INFO is zero and if WANTT is .FALSE., then the contents of H are unspecified on exit. The output state of H in case INF is positive is below under the description of INFO.

LDH Input parameter.

LDH is INTEGER

The leading dimension of the array H. LDH >= max(1, N).

W Output parameter.

W is COMPLEX*16

W is an array, dimension (N). The computed eigenvalues ILO to IHI are stored in the corresponding elements of W. If WANTT is .TRUE., the eigenvalues are stored in the same order as on the diagonal of the Schur form returned in H, with W(i) = H(i,i).

ILOZ Input parameter.

ILOZ is INTEGER

IHIZ Input parameter.

IHIZ is INTEGER

Specify the rows of Z to which transformations must be applied if WANTZ is .TRUE.. 1 <= ILOZ <= ILO; IHI <= IHIZ <= N.

Z Input and output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, N). If WANTZ is .TRUE., on entry Z must contain the current matrix Z of transformations accumulated by CHSEQR, and on exit Z has been updated; transformations are applied only to the submatrix Z(ILOZ:IHIZ,ILO:IHI). If WANTZ is .FALSE., Z is not referenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit .GT. 0: if INFO = i, ZLAHQR failed to compute all the eigenvalues ILO to IHI in a total of 30 iterations per eigenvalue; elements i+1:ihi of W contain those eigenvalues which have been successfully computed.

If INFO .GT. 0 and WANTT is .FALSE., then on exit, the remaining unconverged eigenvalues are the eigenvalues of the upper Hessenberg matrix rows and columns ILO thorugh INFO of the final, output value of H.

If INFO .GT. 0 and WANTT is .TRUE., then on exit (*) (initial value of H)*U = U*(final value of H) where U is an orthognal matrix. The final value of H is upper Hessenberg and triangular in rows and columns INFO+1 through IHI.

If INFO .GT. 0 and WANTZ is .TRUE., then on exit (final value of Z) = (initial value of Z)*U where U is the orthogonal matrix in (*) (regardless of the value of WANTT.)

Related Information

For this routine in other precisions, please see *clahqr*, *dlahqr* and *slahqr*.

4.17.699 zlahr2

zlahr2 reduces the first NB columns of A complex general n-BY-(n-k+1) matrix A so that elements below the k-th subdiagonal are zero. The reduction is performed by an unitary similarity transformation Q^H * A * Q. The routine returns the matrices V and T which determine Q as a block reflector I - V*T*V^H , and also the matrix Y = A * V * T.

This is an auxiliary routine called by ZGEHRD.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlahr2(N, K, NB, A, LDA, TAU, T, LDT, Y, LDY)
```

C specification:

```
#include "armpl.h"

void zlahr2_(const armpl_int_t *n, const armpl_int_t *k,
             const armpl_int_t *nb, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_doublecomplex_t *tau,
             armpl_doublecomplex_t *t, const armpl_int_t *ldt,
             armpl_doublecomplex_t *y, const armpl_int_t *ldy);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix A.

K Input parameter.

K is INTEGER

The offset for the reduction. Elements below the k-th subdiagonal in the first NB columns are reduced to zero. K < N.

NB Input parameter.

NB is INTEGER

The number of columns to be reduced.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA,N-K+1). On entry, the n-by-(n-k+1) general matrix A. On exit, the elements on and above the k-th subdiagonal in the first NB columns are overwritten with the corresponding elements of the reduced matrix; the elements below the k-th subdiagonal, with the array TAU, represent the matrix Q as a product of elementary reflectors. The other columns of A are unchanged. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

TAU Output parameter.

TAU is COMPLEX*16

TAU is an array, dimension (NB). The scalar factors of the elementary reflectors. See Further Details.

T Output parameter.

T is COMPLEX*16

T is an array, dimension (LDT, NB). The upper triangular matrix T.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= NB.

Y Output parameter.

Y is COMPLEX*16

Y is an array, dimension (LDY, NB). The n-by-nb matrix Y.

LDY Input parameter.

LDY is INTEGER

The leading dimension of the array Y. LDY >= N.

Related Information

For this routine in other precisions, please see *clahr2*, *dlahr2* and *slahr2*.

4.17.700 zlaic1

zlaic1 applies one step of incremental condition estimation in its simplest version:

Let x, twonorm(x) = 1, be an approximate singular vector of an j-by-j lower triangular matrix L, such that

```
twonorm(L*x) = sest
```

Then zlaic1 computes sestpr, s, c such that the vector

```
       [ s*x ]
xhat = [  c  ]
```

is an approximate singular vector of

```
       [ L       0   ]
Lhat = [ w**H gamma ]
```

in the sense that

```
twonorm(Lhat*xhat) = sestpr.
```

Depending on JOB, an estimate for the largest or smallest singular value is computed.

Note that [s c]^H and sestpr**2 is an eigenpair of the system

```
diag(sest*sest, 0) + [alpha  gamma] * [ conjg(alpha) ]
                                      [ conjg(gamma) ]
```

where alpha = x^H * w.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlaic1(JOB, J, X, SEST, W, GAMMA, SESTPR, S, C)
```

C specification:

```
#include "armpl.h"

void zlaic1_(const armpl_int_t *job, const armpl_int_t *j,
             const armpl_doublecomplex_t *x, const double *sest,
             const armpl_doublecomplex_t *w,
             const armpl_doublecomplex_t *gamma, double *sestpr,
             armpl_doublecomplex_t *s, armpl_doublecomplex_t *c);
```

Parameters

JOB Input parameter.

JOB is INTEGER

= 1: an estimate for the largest singular value is computed. = 2: an estimate for the smallest singular value is computed.

J Input parameter.

J is INTEGER

Length of X and W

X Input parameter.

X is COMPLEX*16

X is an array, dimension (J). The j-vector x.

SEST Input parameter.

SEST is DOUBLE PRECISION

Estimated singular value of j by j matrix L

W Input parameter.

W is COMPLEX*16

W is an array, dimension (J). The j-vector w.

GAMMA Input parameter.

GAMMA is COMPLEX*16

The diagonal element gamma.

SESTPR Output parameter.

SESTPR is DOUBLE PRECISION

Estimated singular value of (j+1) by (j+1) matrix Lhat.

S Output parameter.

S is COMPLEX*16

Sine needed in forming xhat.

C Output parameter.

C is COMPLEX*16

Cosine needed in forming xhat.

Related Information

For this routine in other precisions, please see *claic1*, *dlaic1* and *slaic1*.

4.17.701 zlals0

zlals0 applies back the multiplying factors of either the left or the right singular vector matrix of a diagonal matrix appended by a row to the right hand side matrix B in solving the least squares problem using the divide-and-conquer SVD approach.

For the left singular vector matrix, three types of orthogonal matrices are involved:

(1L) Givens rotations: the number of such rotations is GIVPTR; the

```
pairs of columns/rows they were applied to are stored in GIVCOL;
and the C- and S-values of these rotations are stored in GIVNUM.
```

(2L) Permutation. The (NL+1)-st row of B is to be moved to the first

```
row, and for J=2:N, PERM(J)-th row of B is to be moved to the
J-th row.
```

(3L) The left singular vector matrix of the remaining matrix.

For the right singular vector matrix, four types of orthogonal matrices are involved:

(1R) The right singular vector matrix of the remaining matrix.

(2R) If SQRE = 1, one extra Givens rotation to generate the right

```
null space.
```

(3R) The inverse transformation of (2L).

(4R) The inverse transformation of (1L).

Syntax

Fortran specification:

```
use armpl_library

subroutine zlals0(ICOMPQ, NL, NR, SQRE, NRHS, B, LDB, BX, LDBX, PERM, GIVPTR,
                  GIVCOL, LDGCOL, GIVNUM, LDGNUM, POLES, DIFL, DIFR, Z, K, C,
                  S, RWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zlals0_(const armpl_int_t *icompq, const armpl_int_t *nl,
             const armpl_int_t *nr, const armpl_int_t *sqre,
             const armpl_int_t *nrhs, armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, armpl_doublecomplex_t *bx,
             const armpl_int_t *ldbx, const armpl_int_t *perm,
             const armpl_int_t *givptr, const armpl_int_t *givcol,
             const armpl_int_t *ldgcol, const double *givnum,
             const armpl_int_t *ldgnum, const double *poles,
             const double *difl, const double *difr, const double *z,
             const armpl_int_t *k, const double *c, const double *s,
             double *rwork, armpl_int_t *info);
```

Parameters

ICOMPQ Input parameter.

ICOMPQ is INTEGER

Specifies whether singular vectors are to be computed in factored form: = 0: Left singular vector matrix. = 1: Right singular vector matrix.

NL Input parameter.

NL is INTEGER

The row dimension of the upper block. NL >= 1.

NR Input parameter.

NR is INTEGER

The row dimension of the lower block. NR >= 1.

SQRE Input parameter.

SQRE is INTEGER

= 0: the lower block is an NR-by-NR square matrix. = 1: the lower block is an NR-by-(NR+1) rectangular matrix.

The bidiagonal matrix has row dimension N = NL + NR + 1, and column dimension M = N + SQRE.

NRHS Input parameter.

NRHS is INTEGER

The number of columns of B and BX. NRHS must be at least 1.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On input, B contains the right hand sides of the least squares problem in rows 1 through M. On output, B contains the solution X in rows 1 through N.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B. LDB must be at least max(1,MAX(M, N)).

BX Output parameter.

BX is COMPLEX*16

BX is an array, dimension (LDBX, NRHS) .

LDBX Input parameter.

LDBX is INTEGER

The leading dimension of BX.

PERM Input parameter.

PERM is INTEGER array, dimension (N)

The permutations (from deflation and sorting) applied to the two blocks.

GIVPTR Input parameter.

GIVPTR is INTEGER

The number of Givens rotations which took place in this subproblem.

GIVCOL Input parameter.

GIVCOL is INTEGER array, dimension (LDGCOL, 2)

Each pair of numbers indicates a pair of rows/columns involved in a Givens rotation.

LDGCOL Input parameter.

LDGCOL is INTEGER

The leading dimension of GIVCOL, must be at least N.

GIVNUM Input parameter.

GIVNUM is DOUBLE PRECISION

GIVNUM is an array, dimension (LDGNUM, 2). Each number indicates the C or S value used in the corresponding Givens rotation.

LDGNUM Input parameter.

LDGNUM is INTEGER

The leading dimension of arrays DIFR, POLES and GIVNUM, must be at least K.

POLES Input parameter.

POLES is DOUBLE PRECISION

POLES is an array, dimension (LDGNUM, 2). On entry, POLES(1:K, 1) contains the new singular values obtained from solving the secular equation, and POLES(1:K, 2) is an array containing the poles in the secular equation.

DIFL Input parameter.

DIFL is DOUBLE PRECISION

DIFL is an array, dimension (K).. On entry, DIFL(I) is the distance between I-th updated (undeflated) singular value and the I-th (undeflated) old singular value.

DIFR Input parameter.

DIFR is DOUBLE PRECISION

DIFR is an array, dimension (LDGNUM, 2).. On entry, DIFR(I, 1) contains the distances between I-th updated (undeflated) singular value and the I+1-th (undeflated) old singular value. And DIFR(I, 2) is the normalizing factor for the I-th right singular vector.

Z Input parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (K). Contain the components of the deflation-adjusted updating row vector.

K Input parameter.

K is INTEGER

Contains the dimension of the non-deflated matrix, This is the order of the related secular equation. 1 <= K <=N.

C Input parameter.

C is DOUBLE PRECISION

C contains garbage if SQRE =0 and the C-value of a Givens rotation related to the right null space if SQRE = 1.

S Input parameter.

S is DOUBLE PRECISION

S contains garbage if SQRE =0 and the S-value of a Givens rotation related to the right null space if SQRE = 1.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension. (K*(1+NRHS) + 2* NRHS)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *clals0*, *dlals0* and *slals0*.

4.17.702 zlalsa

`zlalsa` is an itermediate step in solving the least squares problem by computing the SVD of the coefficient matrix in compact form (The singular vectors are computed as products of simple orthorgonal matrices.).

If ICOMPQ = 0, `zlalsa` applies the inverse of the left singular vector matrix of an upper bidiagonal matrix to the right hand side; and if ICOMPQ = 1, `zlalsa` applies the right singular vector matrix to the right hand side. The singular vector matrices were generated in compact form by `zlalsa`.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlalsa(ICOMPQ, SMLSIZ, N, NRHS, B, LDB, BX, LDBX, U, LDU, VT, K,
                  DIFL, DIFR, Z, POLES, GIVPTR, GIVCOL, LDGCOL, PERM, GIVNUM,
                  C, S, RWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zlalsa_(const armpl_int_t *icompq, const armpl_int_t *smlsiz,
             const armpl_int_t *n, const armpl_int_t *nrhs,
             armpl_doublecomplex_t *b, const armpl_int_t *ldb,
             armpl_doublecomplex_t *bx, const armpl_int_t *ldbx,
             const double *u, const armpl_int_t *ldu, const double *vt,
             const armpl_int_t *k, const double *difl, const double *difr,
             const double *z, const double *poles, const armpl_int_t *givptr,
             const armpl_int_t *givcol, const armpl_int_t *ldgcol,
             const armpl_int_t *perm, const double *givnum, const double *c,
             const double *s, double *rwork, armpl_int_t *iwork,
             armpl_int_t *info);
```

Parameters

ICOMPQ Input parameter.

`ICOMPQ` is `INTEGER`

Specifies whether the left or the right singular vector matrix is involved. = 0: Left singular vector matrix = 1: Right singular vector matrix

SMLSIZ Input parameter.

`SMLSIZ` is `INTEGER`

The maximum size of the subproblems at the bottom of the computation tree.

N Input parameter.

`N` is `INTEGER`

The row and column dimensions of the upper bidiagonal matrix.

NRHS Input parameter.

`NRHS` is `INTEGER`

The number of columns of `B` and `BX`. `NRHS` must be at least 1.

B Input and output parameter.

`B` is `COMPLEX*16`

B is an array, dimension (LDB, NRHS). On input, B contains the right hand sides of the least squares problem in rows 1 through M. On output, B contains the solution X in rows 1 through N.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B in the calling subprogram. LDB must be at least max(1,MAX(M, N)).

BX Output parameter.

BX is COMPLEX*16

BX is an array, dimension (LDBX, NRHS). On exit, the result of applying the left or right singular vector matrix to B.

LDBX Input parameter.

LDBX is INTEGER

The leading dimension of BX.

U Input parameter.

U is DOUBLE PRECISION

U is an array, dimension (LDU, SMLSIZ).. On entry, U contains the left singular vector matrices of all subproblems at the bottom level.

LDU Input parameter.

LDU is INTEGER, LDU = > N.

The leading dimension of arrays U, VT, DIFL, DIFR, POLES, GIVNUM, and Z.

VT Input parameter.

VT is DOUBLE PRECISION

VT is an array, dimension (LDU, SMLSIZ+1).. On entry, VT^H contains the right singular vector matrices of all subproblems at the bottom level.

K Input parameter.

K is INTEGER array, dimension (N).

DIFL Input parameter.

DIFL is DOUBLE PRECISION

DIFL is an array, dimension (LDU, NLVL).. where NLVL = INT(log_2 (N/(SMLSIZ+1))) + 1.

DIFR Input parameter.

DIFR is DOUBLE PRECISION

DIFR is an array, dimension (LDU, 2 * NLVL).. On entry, DIFL(*, I) and DIFR(*, 2 * I -1) record distances between singular values on the I-th level and singular values on the (I -1)-th level, and DIFR(*, 2 * I) record the normalizing factors of the right singular vectors matrices of subproblems on I-th level.

Z Input parameter.

Z is DOUBLE PRECISION

Z is an array, dimension (LDU, NLVL).. On entry, Z(1, I) contains the components of the deflation- adjusted updating row vector for subproblems on the I-th level.

POLES Input parameter.

POLES is DOUBLE PRECISION

POLES is an array, dimension (LDU, 2 * NLVL).. On entry, POLES(*, 2 * I -1: 2 * I) contains the new and old singular values involved in the secular equations on the I-th level.

GIVPTR Input parameter.

GIVPTR is INTEGER array, dimension (N).

On entry, GIVPTR(I) records the number of Givens rotations performed on the I-th problem on the computation tree.

GIVCOL Input parameter.

GIVCOL is INTEGER array, dimension (LDGCOL, 2 * NLVL).

On entry, for each I, GIVCOL(*, 2 * I - 1: 2 * I) records the locations of Givens rotations performed on the I-th level on the computation tree.

LDGCOL Input parameter.

LDGCOL is INTEGER, LDGCOL = > N.

The leading dimension of arrays GIVCOL and PERM.

PERM Input parameter.

PERM is INTEGER array, dimension (LDGCOL, NLVL).

On entry, PERM(*, I) records permutations done on the I-th level of the computation tree.

GIVNUM Input parameter.

GIVNUM is DOUBLE PRECISION

GIVNUM is an array, dimension (LDU, 2 * NLVL).. On entry, GIVNUM(*, 2 *I -1 : 2 * I) records the C- and S- values of Givens rotations performed on the I-th level on the computation tree.

C Input parameter.

C is DOUBLE PRECISION

C is an array, dimension (N).. On entry, if the I-th subproblem is not square, C(I) contains the C-value of a Givens rotation related to the right null space of the I-th subproblem.

S Input parameter.

S is DOUBLE PRECISION

S is an array, dimension (N).. On entry, if the I-th subproblem is not square, S(I) contains the S-value of a Givens rotation related to the right null space of the I-th subproblem.

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension at least. MAX((SMLSZ+1)*NRHS*3, N*(1+NRHS) + 2* NRHS).

IWORK Output parameter.

IWORK is INTEGER array, dimension (3*N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *clalsa*, *dlalsa* and *slalsa*.

4.17.703 zlalsd

`zlalsd` uses the singular value decomposition of A to solve the least squares problem of finding X to minimize the Euclidean norm of each column of A*X-B, where A is N-by-N upper bidiagonal, and X and B are N-by-NRHS. The solution X overwrites B.

The singular values of A smaller than RCOND times the largest singular value are treated as zero in solving the least squares problem; in this case a minimum norm solution is returned. The actual singular values are returned in D in ascending order.

This code makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray XMP, Cray YMP, Cray C 90, or Cray 2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlalsd(UPLO, SMLSIZ, N, NRHS, D, E, B, LDB, RCOND, RANK, WORK,
                  RWORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zlalsd_(const char *uplo, const armpl_int_t *smlsiz,
             const armpl_int_t *n, const armpl_int_t *nrhs, double *d,
             double *e, armpl_doublecomplex_t *b, const armpl_int_t *ldb,
             const double *rcond, armpl_int_t *rank,
             armpl_doublecomplex_t *work, double *rwork, armpl_int_t *iwork,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': D and E define an upper bidiagonal matrix. = 'L': D and E define a lower bidiagonal matrix.

SMLSIZ Input parameter.

SMLSIZ is INTEGER

The maximum size of the subproblems at the bottom of the computation tree.

N Input parameter.

N is INTEGER

The dimension of the bidiagonal matrix. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of columns of B. NRHS must be at least 1.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry D contains the main diagonal of the bidiagonal matrix. On exit, if INFO = 0, D contains its singular values.

E Input and output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N-1). Contains the super-diagonal entries of the bidiagonal matrix. On exit, E has been destroyed.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On input, B contains the right hand sides of the least squares problem. On output, B contains the solution X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of B in the calling subprogram. LDB must be at least max(1, N).

RCOND Input parameter.

RCOND is DOUBLE PRECISION

The singular values of A less than or equal to RCOND times the largest singular value are treated as zero in solving the least squares problem. If RCOND is negative, machine precision is used instead. For example, if diag(S)*X=B were the least squares problem, where diag(S) is a diagonal matrix of singular values, the solution would be X(i) = B(i) / S(i) if S(i) is greater than RCOND*max(S), and X(i) = 0 if S(i) is less than or equal to RCOND*max(S).

RANK Output parameter.

RANK is INTEGER

The number of singular values of A greater than RCOND times the largest singular value.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N * NRHS) .

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension at least. (9* N + 2*N* SMLSIZ + 8*N*NLVL + 3*SMLSIZ* NRHS + MAX((SMLSIZ+1)**2, N*(1+NRHS) + 2* NRHS), where NLVL = MAX(0, INT(LOG_2(MIN(M,N)/(SMLSIZ+1))) + 1)

IWORK Output parameter.

IWORK is INTEGER array, dimension at least

(3*N*NLVL + 11*N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value. > 0: The algorithm failed to compute a singular value while working on the submatrix lying in rows and columns INFO/(N+1) through MOD(INFO,N+1).

Related Information

For this routine in other precisions, please see *clalsd*, *dlalsd* and *slalsd*.

4.17.704 zlamswlq

ZLAMQRTS overwrites the general real M-by-N matrix C with SIDE = 'L' SIDE = 'R' TRANS = 'N': Q * C C * Q TRANS = 'C': Q^H * C C * Q^H where Q is a real orthogonal matrix defined as the product of blocked elementary reflectors computed by short wide LQ factorization (ZLASWLQ)

Syntax

Fortran specification:

```
use armpl_library

subroutine zlamswlq(SIDE, TRANS, M, N, K, MB, NB, A, LDA, T, LDT, C, LDC,
                    WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zlamswlq_(const char *side, const char *trans, const armpl_int_t *m,
               const armpl_int_t *n, const armpl_int_t *k,
               const armpl_int_t *mb, const armpl_int_t *nb,
               armpl_doublecomplex_t *a, const armpl_int_t *lda,
               const armpl_doublecomplex_t *t, const armpl_int_t *ldt,
               armpl_doublecomplex_t *c, const armpl_int_t *ldc,
               armpl_doublecomplex_t *work, const armpl_int_t *lwork,
               armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left; = 'R': apply Q or Q^H from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'C': Conjugate Transpose, apply Q^H .

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >=0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= M.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. M >= K >= 0;

MB Input parameter.

MB is INTEGER

The row block size to be used in the blocked QR. M >= MB >= 1

NB Input parameter.

NB is INTEGER

The column block size to be used in the blocked QR. NB > M.

NB Input parameter.

NB is INTEGER

The block size to be used in the blocked QR. MB > M.

A Input parameter.

A is COMPLEX*16

A is an array, dimension. (LDA, M) if SIDE = 'L', (LDA, N) if SIDE = 'R' The i-th row must contain the vector which defines the blocked elementary reflector H(i), for i = 1,2,...,k, as returned by ZLASWLQ in the first k rows of its array argument A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDA >= max(1, M); if SIDE = 'R', LDA >= max(1, N).

T Input parameter.

T is COMPLEX*16

T is an array, dimension. (M * Number of blocks(CEIL(N-K/NB-K)), The blocked upper triangular block reflectors stored in compact form as a sequence of upper triangular blocks. See below for further details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= MB.

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^H * C or C*Q^H or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

(workspace) COMPLEX*16

(workspace) is an array, dimension (MAX(1, LWORK)) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. If SIDE = 'L', LWORK >= max(1, NB) * MB; if SIDE = 'R', LWORK >= max(1, M) * MB. If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *clamswlq*, *dlamswlq* and *slamswlq*.

4.17.705 zlamtsqr

ZLAMTSQR overwrites the general complex M-by-N matrix C with SIDE = 'L' SIDE = 'R' TRANS = 'N': Q * C C * Q TRANS = 'C': Q^H * C C * Q^H where Q is a real orthogonal matrix defined as the product of blocked elementary reflectors computed by tall skinny QR factorization (ZLATSQR)

Syntax

Fortran specification:

```
use armpl_library

subroutine zlamtsqr(SIDE, TRANS, M, N, K, MB, NB, A, LDA, T, LDT, C, LDC,
                    WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zlamtsqr_(const char *side, const char *trans, const armpl_int_t *m,
               const armpl_int_t *n, const armpl_int_t *k,
               const armpl_int_t *mb, const armpl_int_t *nb,
               armpl_doublecomplex_t *a, const armpl_int_t *lda,
               const armpl_doublecomplex_t *t, const armpl_int_t *ldt,
               armpl_doublecomplex_t *c, const armpl_int_t *ldc,
               armpl_doublecomplex_t *work, const armpl_int_t *lwork,
               armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left; = 'R': apply Q or Q^H from the Right.

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': No transpose, apply Q; = 'C': Conjugate Transpose, apply Q^H .

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >=0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. M >= N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. N >= K >= 0;

MB Input parameter.

MB is INTEGER

The block size to be used in the blocked QR. MB > N. (must be the same as DLATSQR)

NB Input parameter.

NB is INTEGER

The column block size to be used in the blocked QR. N >= NB >= 1.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, K). The i-th column must contain the vector which defines the blockedelementary reflector H(i), for i = 1,2,...,k, as returned by DLATSQR in the first k columns of its array argument A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDA >= max(1, M); if SIDE = 'R', LDA >= max(1, N).

T Input parameter.

T is COMPLEX*16

T is an array, dimension. (N * Number of blocks(CEIL(M-K/MB-K)), The blocked upper triangular block reflectors stored in compact form as a sequence of upper triangular blocks. See below for further details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= NB.

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by Q* C or Q^H * C or C*Q^H or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

(workspace) COMPLEX*16

(workspace) is an array, dimension (MAX(1, LWORK)) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK.

If SIDE = 'L', LWORK >= max(1, N)*NB; if SIDE = 'R', LWORK >= max(1, MB)*NB. If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *clamtsqr*, *dlamtsqr* and *slamtsqr*.

4.17.706 zlangb

`zlangb` returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of an n by n band matrix A, with kl sub-diagonals and ku super-diagonals.

Syntax

Fortran specification:

```
use armpl_library

double precision function zlangb(NORM, N, KL, KU, AB, LDAB, WORK)
```

C specification:

```
#include "armpl.h"

double zlangb_(const char *norm, const armpl_int_t *n, const armpl_int_t *kl,
               const armpl_int_t *ku, const armpl_doublecomplex_t *ab,
               const armpl_int_t *ldab, double *work, ... );
```

Returns

ZLANGB = (max(abs(A(i,j))), NORM = ‘M’ or ‘m’

((norm1(A), NORM = ‘1’, ‘O’ or ‘o’ ((normI(A), NORM = ‘I’ or ‘i’ ((normF(A), NORM = ‘F’, ‘f’, ‘E’ or ‘e’

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

`NORM` is `CHARACTER*1`

Specifies the value to be returned in ZLANGB as described above.

N Input parameter.

`N` is `INTEGER`

The order of the matrix A. `N` >= 0. When `N` = 0, ZLANGB is set to zero.

KL Input parameter.

`KL` is `INTEGER`

The number of sub-diagonals of the matrix A. `KL` >= 0.

KU Input parameter.

`KU` is `INTEGER`

The number of super-diagonals of the matrix A. `KU` >= 0.

AB Input parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). The band matrix A, stored in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(ku+1+i-j,j) = A(i,j) for max(1,j-ku)<=i<=min(n,j+kl).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KL+KU+1.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= N when NORM = 'I'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *clangb*, *dlangb* and *slangb*.

4.17.707 zlange

zlange returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a complex matrix A.

Syntax

Fortran specification:

```
use armpl_library

double precision function zlange(NORM, M, N, A, LDA, WORK)
```

C specification:

```
#include "armpl.h"

double zlange_(const char *norm, const armpl_int_t *m, const armpl_int_t *n,
               const armpl_doublecomplex_t *a, const armpl_int_t *lda,
               double *work, ... );
```

Returns

ZLANGE = (max(abs(A(i,j))), NORM = 'M' or 'm'

((norm1(A), NORM = '1', 'O' or 'o' ((normI(A), NORM = 'I' or 'i' ((normF(A), NORM = 'F', 'f', 'E' or 'e'

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in ZLANGE as described above.

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0. When M = 0, ZLANGE is set to zero.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0. When N = 0, ZLANGE is set to zero.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The m by n matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(M,1).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= M when NORM = 'I'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *clange*, *dlange* and *slange*. It also exists with a native C interface as *LAPACKE_zlange*.

4.17.708 zlangt

zlangt returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a complex tridiagonal matrix A.

Syntax

Fortran specification:

```
use armpl_library

double precision function zlangt(NORM, N, DL, D, DU)
```

C specification:

```
#include "armpl.h"

double zlangt_(const char *norm, const armpl_int_t *n,
               const armpl_doublecomplex_t *dl,
               const armpl_doublecomplex_t *d,
               const armpl_doublecomplex_t *du, ... );
```

Returns

ZLANGT = (max(abs(A(i,j))), NORM = 'M' or 'm'

((norm1(A), NORM = '1', 'O' or 'o' ((normI(A), NORM = 'I' or 'i' ((normF(A), NORM = 'F', 'f', 'E' or 'e'

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in ZLANGT as described above.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0. When N = 0, ZLANGT is set to zero.

DL Input parameter.

DL is COMPLEX*16

DL is an array, dimension (N-1). The (n-1) sub-diagonal elements of A.

D Input parameter.

D is COMPLEX*16

D is an array, dimension (N). The diagonal elements of A.

DU Input parameter.

DU is COMPLEX*16

DU is an array, dimension (N-1). The (n-1) super-diagonal elements of A.

Related Information

For this routine in other precisions, please see *clangt*, *dlangt* and *slangt*.

4.17.709 zlanhb

zlanhb returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of an n by n hermitian band matrix A, with k super-diagonals.

Syntax

Fortran specification:

```
use armpl_library

double precision function zlanhb(NORM, UPLO, N, K, AB, LDAB, WORK)
```

C specification:

```
#include "armpl.h"

double zlanhb_(const char *norm, const char *uplo, const armpl_int_t *n,
               const armpl_int_t *k, const armpl_doublecomplex_t *ab,
               const armpl_int_t *ldab, double *work, ... );
```

Returns

ZLANHB = (max(abs(A(i,j))), NORM = ‘M’ or ‘m’

((norm1(A), NORM = ‘1’, ‘O’ or ‘o’ ((normI(A), NORM = ‘I’ or ‘i’ ((normF(A), NORM = ‘F’, ‘f’, ‘E’ or ‘e’

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in ZLANHB as described above.

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the band matrix A is supplied. = ‘U’: Upper triangular = ‘L’: Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0. When N = 0, ZLANHB is set to zero.

K Input parameter.

K is INTEGER

The number of super-diagonals or sub-diagonals of the band matrix A. K >= 0.

AB Input parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). The upper or lower triangle of the hermitian band matrix A, stored in the first K+1 rows of AB. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = ‘U’, AB(k+1+i-j,j) = A(i,j) for max(1,j-k)<=i<=j; if UPLO = ‘L’, AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+k). Note that the imaginary parts of the diagonal elements need not be set and are assumed to be zero.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= K+1.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= N when NORM = ‘I’ or ‘1’ or ‘O’; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *clanhb*.

4.17.710 zlanhe

`zlanhe` returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a complex hermitian matrix A.

Syntax

Fortran specification:

```
use armpl_library

double precision function zlanhe(NORM, UPLO, N, A, LDA, WORK)
```

C specification:

```
#include "armpl.h"

double zlanhe_(const char *norm, const char *uplo, const armpl_int_t *n,
               const armpl_doublecomplex_t *a, const armpl_int_t *lda,
               double *work, ... );
```

Returns

ZLANHE = (max(abs(A(i,j))), NORM = ‘M’ or ‘m’

((norm1(A), NORM = ‘1’, ‘O’ or ‘o’ ((normI(A), NORM = ‘I’ or ‘i’ ((normF(A), NORM = ‘F’, ‘f’, ‘E’ or ‘e’

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

`NORM` is `CHARACTER*1`

Specifies the value to be returned in ZLANHE as described above.

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

Specifies whether the upper or lower triangular part of the hermitian matrix `A` is to be referenced. = ‘`U`’: Upper triangular part of `A` is referenced = ‘`L`’: Lower triangular part of `A` is referenced

N Input parameter.

`N` is `INTEGER`

The order of the matrix `A`. `N` >= 0. When `N` = 0, ZLANHE is set to zero.

A Input parameter.

`A` is `COMPLEX*16`

`A` is an array, dimension (`LDA`, `N`). The hermitian matrix `A`. If `UPLO` = ‘`U`’, the leading n by n upper triangular part of `A` contains the upper triangular part of the matrix `A`, and the strictly lower triangular part of A is not

referenced. If UPLO = 'L', the leading n by n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. Note that the imaginary parts of the diagonal elements need not be set and are assumed to be zero.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(N,1).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= N when NORM = 'I' or '1' or 'O'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *clanhe*. It also exists with a native C interface as *LAPACKE_zlanhe*.

4.17.711 zlanhf

zlanhf returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a complex Hermitian matrix A in RFP format.

Syntax

Fortran specification:

```
use armpl_library

double precision function zlanhf(NORM, TRANSR, UPLO, N, A, WORK)
```

C specification:

```
#include "armpl.h"

double zlanhf_(const char *norm, const char *transr, const char *uplo,
               const armpl_int_t *n, const armpl_doublecomplex_t *a,
               double *work, ... );
```

Returns

ZLANHF = (max(abs(A(i,j))), NORM = 'M' or 'm'

((norm1(A), NORM = '1', 'O' or 'o' ((normI(A), NORM = 'I' or 'i' ((normF(A), NORM = 'F', 'f', 'E' or 'e'

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER

Specifies the value to be returned in ZLANHF as described above.

TRANSR Input parameter.

TRANSR is CHARACTER

Specifies whether the RFP format of A is normal or conjugate-transposed format. = 'N': RFP format is Normal = 'C': RFP format is Conjugate-transposed

UPLO Input parameter.

UPLO is CHARACTER

On entry, UPLO specifies whether the RFP matrix A came from an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' RFP A came from an upper triangular matrix

UPLO = 'L' or 'l' RFP A came from a lower triangular matrix

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0. When N = 0, ZLANHF is set to zero.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (N*(N+1)/2);. On entry, the matrix A in RFP Format. RFP Format is described by TRANSR, UPLO and N as follows: If TRANSR='N' then RFP A is (0:N,0:K-1) when N is even; K=N/2. RFP A is (0:N-1,0:K) when N is odd; K=N/2. If TRANSR = 'C' then RFP is the Conjugate-transpose of RFP A as defined when TRANSR = 'N'. The contents of RFP A are defined by UPLO as follows: If UPLO = 'U' the RFP A contains the (N*(N+1)/2) elements of upper packed A either in normal or conjugate-transpose Format. If UPLO = 'L' the RFP A contains the (N*(N+1) /2) elements of lower packed A either in normal or conjugate-transpose Format. The LDA of RFP A is (N+1)/2 when TRANSR = 'C'. When TRANSR is 'N' the LDA is N+1 when N is even and is N when is odd. See the Note below for more details. Unchanged on exit.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (LWORK),. where LWORK >= N when NORM = 'I' or '1' or 'O'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *clanhf*.

4.17.712 zlanhp

zlanhp returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a complex hermitian matrix A, supplied in packed form.

Syntax

Fortran specification:

```
use armpl_library

double precision function zlanhp(NORM, UPLO, N, AP, WORK)
```

C specification:

```
#include "armpl.h"

double zlanhp_(const char *norm, const char *uplo, const armpl_int_t *n,
               const armpl_doublecomplex_t *ap, double *work, ... );
```

Returns

ZLANHP = (max(abs(A(i,j))), NORM = ‘M’ or ‘m’

((norm1(A), NORM = ‘1’, ‘O’ or ‘o’ ((normI(A), NORM = ‘I’ or ‘i’ ((normF(A), NORM = ‘F’, ‘f’, ‘E’ or ‘e’

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in ZLANHP as described above.

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the hermitian matrix A is supplied. = ‘U’: Upper triangular part of A is supplied = ‘L’: Lower triangular part of A is supplied

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0. When N = 0, ZLANHP is set to zero.

AP Input parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). The upper or lower triangle of the hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = ‘U’, AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = ‘L’, AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n. Note that the imaginary parts of the diagonal elements need not be set and are assumed to be zero.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= N when NORM = ‘I’ or ‘1’ or ‘O’; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *clanhp*.

4.17.713 zlanhs

zlanhs returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a Hessenberg matrix A.

Syntax

Fortran specification:

```
use armpl_library

double precision function zlanhs(NORM, N, A, LDA, WORK)
```

C specification:

```
#include "armpl.h"

double zlanhs_(const char *norm, const armpl_int_t *n,
               const armpl_doublecomplex_t *a, const armpl_int_t *lda,
               double *work, ... );
```

Returns

ZLANHS = (max(abs(A(i,j))), NORM = ‘M’ or ‘m’

((norm1(A), NORM = ‘1’, ‘O’ or ‘o’ ((normI(A), NORM = ‘I’ or ‘i’ ((normF(A), NORM = ‘F’, ‘f’, ‘E’ or ‘e’

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in ZLANHS as described above.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0. When N = 0, ZLANHS is set to zero.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The n by n upper Hessenberg matrix A; the part of A below the first sub-diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(N,1).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= N when NORM = ‘I’; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *clanhs*, *dlanhs* and *slanhs*.

4.17.714 zlanht

`zlanht` returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a complex Hermitian tridiagonal matrix A.

Syntax

Fortran specification:

```
use armpl_library

double precision function zlanht(NORM, N, D, E)
```

C specification:

```
#include "armpl.h"

double zlanht_(const char *norm, const armpl_int_t *n, const double *d,
               const armpl_doublecomplex_t *e, ... );
```

Returns

ZLANHT = (max(abs(A(i,j))), NORM = ‘M’ or ‘m’

((norm1(A), NORM = ‘1’, ‘O’ or ‘o’ ((normI(A), NORM = ‘I’ or ‘i’ ((normF(A), NORM = ‘F’, ‘f’, ‘E’ or ‘e’

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

`NORM` is `CHARACTER*1`

Specifies the value to be returned in ZLANHT as described above.

N Input parameter.

`N` is `INTEGER`

The order of the matrix A. `N` >= 0. When `N` = 0, ZLANHT is set to zero.

D Input parameter.

`D` is `DOUBLE PRECISION`

`D` is an array, dimension (`N`). The diagonal elements of A.

E Input parameter.

`E` is `COMPLEX*16`

`E` is an array, dimension (N-1). The (n-1) sub-diagonal or super-diagonal elements of A.

Related Information

For this routine in other precisions, please see *clanht*.

4.17.715 zlansb

`zlansb` returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of an n by n symmetric band matrix A, with k super-diagonals.

Syntax

Fortran specification:

```
use armpl_library

double precision function zlansb(NORM, UPLO, N, K, AB, LDAB, WORK)
```

C specification:

```
#include "armpl.h"

double zlansb_(const char *norm, const char *uplo, const armpl_int_t *n,
               const armpl_int_t *k, const armpl_doublecomplex_t *ab,
               const armpl_int_t *ldab, double *work, ... );
```

Returns

ZLANSB = (max(abs(A(i,j))), NORM = ‘M’ or ‘m’

((norm1(A), NORM = ‘1’, ‘O’ or ‘o’ ((normI(A), NORM = ‘I’ or ‘i’ ((normF(A), NORM = ‘F’, ‘f’, ‘E’ or ‘e’

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

`NORM` is `CHARACTER*1`

Specifies the value to be returned in ZLANSB as described above.

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

Specifies whether the upper or lower triangular part of the band matrix A is supplied. = ‘`U`’: Upper triangular part is supplied = ‘`L`’: Lower triangular part is supplied

N Input parameter.

`N` is `INTEGER`

The order of the matrix A. `N` >= 0. When `N` = 0, ZLANSB is set to zero.

K Input parameter.

`K` is `INTEGER`

The number of super-diagonals or sub-diagonals of the band matrix A. `K` >= 0.

AB Input parameter.

`AB` is `COMPLEX*16`

`AB` is an array, dimension (`LDAB`, `N`). The upper or lower triangle of the symmetric band matrix A, stored in the first K+1 rows of `AB`. The j-th column of A is stored in the j-th column of the array `AB` as follows: if `UPLO` = ‘`U`’, AB(k+1+i-j,j) = A(i,j) for max(1,j-k)<=i<=j; if `UPLO` = ‘`L`’, AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+k).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= K+1.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= N when NORM = 'I' or '1' or 'O'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *clansb*, *dlansb* and *slansb*.

4.17.716 zlansp

zlansp returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a complex symmetric matrix A, supplied in packed form.

Syntax

Fortran specification:

```
use armpl_library

double precision function zlansp(NORM, UPLO, N, AP, WORK)
```

C specification:

```
#include "armpl.h"

double zlansp_(const char *norm, const char *uplo, const armpl_int_t *n,
               const armpl_doublecomplex_t *ap, double *work, ... );
```

Returns

ZLANSP = (max(abs(A(i,j))), NORM = 'M' or 'm'

((norm1(A), NORM = '1', 'O' or 'o' ((normI(A), NORM = 'I' or 'i' ((normF(A), NORM = 'F', 'f', 'E' or 'e'

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in ZLANSP as described above.

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is supplied. = 'U': Upper triangular part of A is supplied = 'L': Lower triangular part of A is supplied

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0. When N = 0, ZLANSP is set to zero.

AP Input parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). The upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= N when NORM = 'I' or '1' or 'O'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *clansp*, *dlansp* and *slansp*.

4.17.717 zlansy

zlansy returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a complex symmetric matrix A.

Syntax

Fortran specification:

```
use armpl_library

double precision function zlansy(NORM, UPLO, N, A, LDA, WORK)
```

C specification:

```
#include "armpl.h"

double zlansy_(const char *norm, const char *uplo, const armpl_int_t *n,
               const armpl_doublecomplex_t *a, const armpl_int_t *lda,
               double *work, ... );
```

Returns

ZLANSY = (max(abs(A(i,j))), NORM = 'M' or 'm'

((norm1(A), NORM = '1', 'O' or 'o' ((normI(A), NORM = 'I' or 'i' ((normF(A), NORM = 'F', 'f', 'E' or 'e'

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in ZLANSY as described above.

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is to be referenced. = 'U': Upper triangular part of A is referenced = 'L': Lower triangular part of A is referenced

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0. When N = 0, ZLANSY is set to zero.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The symmetric matrix A. If UPLO = 'U', the leading n by n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n by n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(N,1).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= N when NORM = 'I' or '1' or 'O'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *clansy*, *dlansy* and *slansy*. It also exists with a native C interface as *LAPACKE_zlansy*.

4.17.718 zlantb

zlantb returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of an n by n triangular band matrix A, with (k + 1) diagonals.

Syntax

Fortran specification:

```
use armpl_library

double precision function zlantb(NORM, UPLO, DIAG, N, K, AB, LDAB, WORK)
```

C specification:

```
#include "armpl.h"

double zlantb_(const char *norm, const char *uplo, const char *diag,
               const armpl_int_t *n, const armpl_int_t *k,
               const armpl_doublecomplex_t *ab, const armpl_int_t *ldab,
               double *work, ... );
```

Returns

ZLANTB = (max(abs(A(i,j))), NORM = 'M' or 'm'

((norm1(A), NORM = '1', 'O' or 'o' ((normI(A), NORM = 'I' or 'i' ((normF(A), NORM = 'F', 'f', 'E' or 'e'

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in ZLANTB as described above.

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the matrix A is upper or lower triangular. = 'U': Upper triangular = 'L': Lower triangular

DIAG Input parameter.

DIAG is CHARACTER*1

Specifies whether or not the matrix A is unit triangular. = 'N': Non-unit triangular = 'U': Unit triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0. When N = 0, ZLANTB is set to zero.

K Input parameter.

K is INTEGER

The number of super-diagonals of the matrix A if UPLO = 'U', or the number of sub-diagonals of the matrix A if UPLO = 'L'. K >= 0.

AB Input parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). The upper or lower triangular band matrix A, stored in the first k+1 rows of AB. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(k+1+i-j,j) = A(i,j) for max(1,j-k)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+k). Note that when DIAG = 'U', the elements of the array AB corresponding to the diagonal elements of the matrix A are not referenced, but are assumed to be one.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= K+1.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= N when NORM = 'I'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *clantb*, *dlantb* and *slantb*.

4.17.719 zlantp

zlantp returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a triangular matrix A, supplied in packed form.

Syntax

Fortran specification:

```
use armpl_library

double precision function zlantp(NORM, UPLO, DIAG, N, AP, WORK)
```

C specification:

```
#include "armpl.h"

double zlantp_(const char *norm, const char *uplo, const char *diag,
               const armpl_int_t *n, const armpl_doublecomplex_t *ap,
               double *work, ... );
```

Returns

ZLANTP = (max(abs(A(i,j))), NORM = 'M' or 'm'

((norm1(A), NORM = '1', 'O' or 'o' ((normI(A), NORM = 'I' or 'i' ((normF(A), NORM = 'F', 'f', 'E' or 'e'

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in ZLANTP as described above.

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the matrix A is upper or lower triangular. = 'U': Upper triangular = 'L': Lower triangular

DIAG Input parameter.

DIAG is CHARACTER*1

Specifies whether or not the matrix A is unit triangular. = 'N': Non-unit triangular = 'U': Unit triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0. When N = 0, ZLANTP is set to zero.

AP Input parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). The upper or lower triangular matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n. Note that when DIAG = 'U', the elements of the array AP corresponding to the diagonal elements of the matrix A are not referenced, but are assumed to be one.

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= N when NORM = 'I'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *clantp*, *dlantp* and *slantp*.

4.17.720 zlantr

zlantr returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a trapezoidal or triangular matrix A.

Syntax

Fortran specification:

```
use armpl_library

double precision function zlantr(NORM, UPLO, DIAG, M, N, A, LDA, WORK)
```

C specification:

```
#include "armpl.h"

double zlantr_(const char *norm, const char *uplo, const char *diag,
               const armpl_int_t *m, const armpl_int_t *n,
               const armpl_doublecomplex_t *a, const armpl_int_t *lda,
               double *work, ... );
```

Returns

ZLANTR = (max(abs(A(i,j))), NORM = 'M' or 'm'

((norm1(A), NORM = '1', 'O' or 'o' ((normI(A), NORM = 'I' or 'i' ((normF(A), NORM = 'F', 'f', 'E' or 'e'

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

NORM Input parameter.

NORM is CHARACTER*1

Specifies the value to be returned in ZLANTR as described above.

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the matrix A is upper or lower trapezoidal. = 'U': Upper trapezoidal = 'L': Lower trapezoidal Note that A is triangular instead of trapezoidal if M = N.

DIAG Input parameter.

DIAG is CHARACTER*1

Specifies whether or not the matrix A has unit diagonal. = 'N': Non-unit diagonal = 'U': Unit diagonal

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0, and if UPLO = 'U', M <= N. When M = 0, ZLANTR is set to zero.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0, and if UPLO = 'L', N <= M. When N = 0, ZLANTR is set to zero.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The trapezoidal matrix A (A is triangular if M = N). If UPLO = 'U', the leading m by n upper trapezoidal part of the array A contains the upper trapezoidal matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading m by n lower trapezoidal part of the array A contains the lower trapezoidal matrix, and the strictly upper triangular part of A is not referenced. Note that when DIAG = 'U', the diagonal elements of A are not referenced and are assumed to be one.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(M,1).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (MAX(1,LWORK)),. where LWORK >= M when NORM = 'I'; otherwise, WORK is not referenced.

Related Information

For this routine in other precisions, please see *clantr*, *dlantr* and *slantr*. It also exists with a native C interface as *LAPACKE_zlantr*.

4.17.721 zlapll

Given two column vectors X and Y, let

```
A = ( X Y ).
```

The subroutine first computes the QR factorization of A = Q*R, and then computes the SVD of the 2-by-2 upper triangular matrix R. The smaller singular value of R is returned in SSMIN, which is used as the measurement of the linear dependency of the vectors X and Y.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlapll(N, X, INCX, Y, INCY, SSMIN)
```

C specification:

```
#include "armpl.h"

void zlapll_(const armpl_int_t *n, armpl_doublecomplex_t *x,
             const armpl_int_t *incx, armpl_doublecomplex_t *y,
             const armpl_int_t *incy, double *ssmin);
```

Parameters

N Input parameter.

N is INTEGER

The length of the vectors X and Y.

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension (1+(N-1)*INCX). On entry, X contains the N-vector X. On exit, X is overwritten.

INCX Input parameter.

INCX is INTEGER

The increment between successive elements of X. INCX > 0.

Y Input and output parameter.

Y is COMPLEX*16

Y is an array, dimension (1+(N-1)*INCY). On entry, Y contains the N-vector Y. On exit, Y is overwritten.

INCY Input parameter.

INCY is INTEGER

The increment between successive elements of Y. INCY > 0.

SSMIN Output parameter.

SSMIN is DOUBLE PRECISION

The smallest singular value of the N-by-2 matrix A = (X Y).

Related Information

For this routine in other precisions, please see *clapll*, *dlapll* and *slapll*.

4.17.722 zlapmr

`zlapmr` rearranges the rows of the M by N matrix X as specified by the permutation K(1),K(2),...,K(M) of the integers 1,...,M. If FORWRD = .TRUE., forward permutation:

```
X(K(I),*) is moved X(I,*) for I = 1,2,...,M.
```

If FORWRD = .FALSE., backward permutation:

```
X(I,*) is moved to X(K(I),*) for I = 1,2,...,M.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zlapmr(FORWRD, M, N, X, LDX, K)
```

C specification:

```
#include "armpl.h"

void zlapmr_(const armpl_int_t *forwrd, const armpl_int_t *m,
             const armpl_int_t *n, armpl_doublecomplex_t *x,
             const armpl_int_t *ldx, armpl_int_t *k);
```

Parameters

FORWRD Input parameter.

FORWRD is LOGICAL

= .TRUE., forward permutation = .FALSE., backward permutation

M Input parameter.

M is INTEGER

The number of rows of the matrix X. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix X. N >= 0.

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, N). On entry, the M by N matrix X. On exit, X contains the permuted matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X, LDX >= MAX(1, M).

K Input and output parameter.

K is INTEGER array, dimension (M)

On entry, K contains the permutation vector. K is used as internal workspace, but reset to its original value on output.

Related Information

For this routine in other precisions, please see *clapmr*, *dlapmr* and *slapmr*. It also exists with a native C interface as *LAPACKE_zlapmr*.

4.17.723 zlapmt

`zlapmt` rearranges the columns of the M by N matrix X as specified by the permutation K(1),K(2),...,K(N) of the integers 1,...,N. If FORWRD = .TRUE., forward permutation:

```
X(*,K(J)) is moved X(*,J) for J = 1,2,...,N.
```

If FORWRD = .FALSE., backward permutation:

```
X(*,J) is moved to X(*,K(J)) for J = 1,2,...,N.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zlapmt(FORWRD, M, N, X, LDX, K)
```

C specification:

```
#include "armpl.h"

void zlapmt_(const armpl_int_t *forwrd, const armpl_int_t *m,
             const armpl_int_t *n, armpl_doublecomplex_t *x,
             const armpl_int_t *ldx, armpl_int_t *k);
```

Parameters

FORWRD Input parameter.

FORWRD is LOGICAL

= .TRUE., forward permutation = .FALSE., backward permutation

M Input parameter.

M is INTEGER

The number of rows of the matrix X. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix X. N >= 0.

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, N). On entry, the M by N matrix X. On exit, X contains the permuted matrix X.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the array X, LDX >= MAX(1, M).

K Input and output parameter.

K is INTEGER array, dimension (N)

On entry, K contains the permutation vector. K is used as internal workspace, but reset to its original value on output.

Related Information

For this routine in other precisions, please see *clapmt*, *dlapmt* and *slapmt*. It also exists with a native C interface as *LAPACKE_zlapmt*.

4.17.724 zlaqgb

zlaqgb equilibrates a general M by N band matrix A with KL subdiagonals and KU superdiagonals using the row and scaling factors in the vectors R and C.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlaqgb(M, N, KL, KU, AB, LDAB, R, C, ROWCND, COLCND, AMAX, EQUED)
```

C specification:

```
#include "armpl.h"

void zlaqgb_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *kl, const armpl_int_t *ku,
             armpl_doublecomplex_t *ab, const armpl_int_t *ldab,
             const double *r, const double *c, const double *rowcnd,
             const double *colcnd, const double *amax, char *equed, ... );
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

KL Input parameter.

KL is INTEGER

The number of subdiagonals within the band of A. KL >= 0.

KU Input parameter.

KU is INTEGER

The number of superdiagonals within the band of A. KU >= 0.

AB Input and output parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). On entry, the matrix A in band storage, in rows 1 to KL+KU+1. The j-th column of A is stored in the j-th column of the array AB as follows: AB(ku+1+i-j,j) = A(i,j) for max(1,j-ku)<=i<=min(m,j+kl)

On exit, the equilibrated matrix, in the same storage format as A. See EQUED for the form of the equilibrated matrix.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDA >= KL+KU+1.

R Input parameter.

R is DOUBLE PRECISION

R is an array, dimension (M). The row scale factors for A.

C Input parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). The column scale factors for A.

ROWCND Input parameter.

ROWCND is DOUBLE PRECISION

Ratio of the smallest R(i) to the largest R(i).

COLCND Input parameter.

COLCND is DOUBLE PRECISION

Ratio of the smallest C(i) to the largest C(i).

AMAX Input parameter.

AMAX is DOUBLE PRECISION

Absolute value of largest matrix entry.

EQUED Output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration = 'R': Row equilibration, i.e., A has been premultiplied by diag(R). = 'C': Column equilibration, i.e., A has been postmultiplied by diag(C). = 'B': Both row and column equilibration, i.e., A has been replaced by diag(R) * A * diag(C).

Related Information

For this routine in other precisions, please see *claqgb*, *dlaqgb* and *slaqgb*.

4.17.725 zlaqge

zlaqge equilibrates a general M by N matrix A using the row and column scaling factors in the vectors R and C.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlaqge(M, N, A, LDA, R, C, ROWCND, COLCND, AMAX, EQUED)
```

C specification:

```
#include "armpl.h"

void zlaqge_(const armpl_int_t *m, const armpl_int_t *n,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const double *r, const double *c, const double *rowcnd,
             const double *colcnd, const double *amax, char *equed, ... );
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the M by N matrix A. On exit, the equilibrated matrix. See EQUED for the form of the equilibrated matrix.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(M,1).

R Input parameter.

R is DOUBLE PRECISION

R is an array, dimension (M). The row scale factors for A.

C Input parameter.

C is DOUBLE PRECISION

C is an array, dimension (N). The column scale factors for A.

ROWCND Input parameter.

ROWCND is DOUBLE PRECISION

Ratio of the smallest R(i) to the largest R(i).

COLCND Input parameter.

COLCND is DOUBLE PRECISION

Ratio of the smallest C(i) to the largest C(i).

AMAX Input parameter.

AMAX is DOUBLE PRECISION

Absolute value of largest matrix entry.

EQUED Output parameter.

EQUED is CHARACTER*1

Specifies the form of equilibration that was done. = 'N': No equilibration = 'R': Row equilibration, i.e., A has been premultiplied by diag(R). = 'C': Column equilibration, i.e., A has been postmultiplied by diag(C). = 'B': Both row and column equilibration, i.e., A has been replaced by diag(R) * A * diag(C).

Related Information

For this routine in other precisions, please see *claqge*, *dlaqge* and *slaqge*.

4.17.726 zlaqhb

zlaqhb equilibrates a Hermitian band matrix A using the scaling factors in the vector S.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlaqhb(UPLO, N, KD, AB, LDAB, S, SCOND, AMAX, EQUED)
```

C specification:

```
#include "armpl.h"

void zlaqhb_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
             armpl_doublecomplex_t *ab, const armpl_int_t *ldab, double *s,
             const double *scond, const double *amax, char *equed, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored. = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of super-diagonals of the matrix A if UPLO = 'U', or the number of sub-diagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, if INFO = 0, the triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H of the band matrix A, in the same storage format as A.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

S Output parameter.

S is DOUBLE PRECISION

S is an array, dimension (N). The scale factors for A.

SCOND Input parameter.

SCOND is DOUBLE PRECISION

Ratio of the smallest S(i) to the largest S(i).

AMAX Input parameter.

AMAX is DOUBLE PRECISION

Absolute value of largest matrix entry.

EQUED Output parameter.

EQUED is CHARACTER*1

Specifies whether or not equilibration was done. = 'N': No equilibration. = 'Y': Equilibration was done, i.e., A has been replaced by diag(S) * A * diag(S).

Related Information

For this routine in other precisions, please see *claqhb*.

4.17.727 zlaqhe

zlaqhe equilibrates a Hermitian matrix A using the scaling factors in the vector S.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlaqhe(UPLO, N, A, LDA, S, SCOND, AMAX, EQUED)
```

C specification:

```
#include "armpl.h"

void zlaqhe_(const char *uplo, const armpl_int_t *n, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, const double *s, const double *scond,
             const double *amax, char *equed, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the Hermitian matrix A is stored. = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading n by n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n by n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if EQUED = 'Y', the equilibrated matrix: diag(S) * A * diag(S).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(N,1).

S Input parameter.

S is DOUBLE PRECISION

S is an array, dimension (N). The scale factors for A.

SCOND Input parameter.

SCOND is DOUBLE PRECISION

Ratio of the smallest S(i) to the largest S(i).

AMAX Input parameter.

AMAX is DOUBLE PRECISION

Absolute value of largest matrix entry.

EQUED Output parameter.

EQUED is CHARACTER*1

Specifies whether or not equilibration was done. = 'N': No equilibration. = 'Y': Equilibration was done, i.e., A has been replaced by diag(S) * A * diag(S).

Related Information

For this routine in other precisions, please see *claqhe*.

4.17.728 zlaqhp

zlaqhp equilibrates a Hermitian matrix A using the scaling factors in the vector S.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlaqhp(UPLO, N, AP, S, SCOND, AMAX, EQUED)
```

C specification:

```
#include "armpl.h"

void zlaqhp_(const char *uplo, const armpl_int_t *n,
             armpl_doublecomplex_t *ap, const double *s, const double *scond,
             const double *amax, char *equed, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the Hermitian matrix A is stored. = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input and output parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the Hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

On exit, the equilibrated matrix: diag(S) * A * diag(S), in the same storage format as A.

S Input parameter.

S is DOUBLE PRECISION

S is an array, dimension (N). The scale factors for A.

SCOND Input parameter.

SCOND is DOUBLE PRECISION

Ratio of the smallest S(i) to the largest S(i).

AMAX Input parameter.

AMAX is DOUBLE PRECISION

Absolute value of largest matrix entry.

EQUED Output parameter.

EQUED is CHARACTER*1

Specifies whether or not equilibration was done. = 'N': No equilibration. = 'Y': Equilibration was done, i.e., A has been replaced by diag(S) * A * diag(S).

Related Information

For this routine in other precisions, please see *claqhp*.

4.17.729 zlaqp2

`zlaqp2` computes a QR factorization with column pivoting of the block A(OFFSET+1:M,1:N). The block A(1:OFFSET,1:N) is accordingly pivoted, but not factorized.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlaqp2(M, N, OFFSET, A, LDA, JPVT, TAU, VN1, VN2, WORK)
```

C specification:

```
#include "armpl.h"

void zlaqp2_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *offset, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *jpvt,
             armpl_doublecomplex_t *tau, double *vn1, double *vn2,
             armpl_doublecomplex_t *work);
```

Parameters

M Input parameter.

`M` is `INTEGER`

The number of rows of the matrix `A`. `M` >= 0.

N Input parameter.

`N` is `INTEGER`

The number of columns of the matrix `A`. `N` >= 0.

OFFSET Input parameter.

`OFFSET` is `INTEGER`

The number of rows of the matrix `A` that must be pivoted but no factorized. `OFFSET` >= 0.

A Input and output parameter.

`A` is `COMPLEX*16`

`A` is an array, dimension (`LDA`, `N`). On entry, the M-by-N matrix `A`. On exit, the upper triangle of block A(OFFSET+1:M,1:N) is the triangular factor obtained; the elements in block A(OFFSET+1:M,1:N) below the diagonal, together with the array `TAU`, represent the orthogonal matrix Q as a product of elementary reflectors. Block A(1:OFFSET,1:N) has been accordingly pivoted, but no factorized.

LDA Input parameter.

`LDA` is `INTEGER`

The leading dimension of the array `A`. `LDA` >= max(1, `M`).

JPVT Input and output parameter.

JPVT is INTEGER array, dimension (N)

On entry, if JPVT(i) .ne. 0, the i-th column of A is permuted to the front of A*P (a leading column); if JPVT(i) = 0, the i-th column of A is a free column. On exit, if JPVT(i) = k, then the i-th column of A*P was the k-th column of A.

TAU Output parameter.

TAU is COMPLEX*16

TAU is an array, dimension (min(M, N)). The scalar factors of the elementary reflectors.

VN1 Input and output parameter.

VN1 is DOUBLE PRECISION

VN1 is an array, dimension (N). The vector with the partial column norms.

VN2 Input and output parameter.

VN2 is DOUBLE PRECISION

VN2 is an array, dimension (N). The vector with the exact column norms.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N) .

Related Information

For this routine in other precisions, please see *claqp2*, *dlaqp2* and *slaqp2*.

4.17.730 zlaqps

zlaqps computes a step of QR factorization with column pivoting of a complex M-by-N matrix A by using Blas-3. It tries to factorize NB columns from A starting from the row OFFSET+1, and updates all of the matrix with Blas-3 xGEMM.

In some cases, due to catastrophic cancellations, it cannot factorize NB columns. Hence, the actual number of factorized columns is returned in KB.

Block A(1:OFFSET,1:N) is accordingly pivoted, but not factorized.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlaqps(M, N, OFFSET, NB, KB, A, LDA, JPVT, TAU, VN1, VN2, AUXV, F,
                  LDF)
```

C specification:

```
#include "armpl.h"

void zlaqps_(const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *offset, const armpl_int_t *nb,
             armpl_int_t *kb, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *jpvt,
```

(continues on next page)

(continued from previous page)

```
                armpl_doublecomplex_t *tau, double *vn1, double *vn2,
                armpl_doublecomplex_t *auxv, armpl_doublecomplex_t *f,
                const armpl_int_t *ldf);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0

OFFSET Input parameter.

OFFSET is INTEGER

The number of rows of A that have been factorized in previous steps.

NB Input parameter.

NB is INTEGER

The number of columns to factorize.

KB Output parameter.

KB is INTEGER

The number of columns actually factorized.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, block A(OFFSET+1:M,1:KB) is the triangular factor obtained and block A(1:OFFSET,1:N) has been accordingly pivoted, but no factorized. The rest of the matrix, block A(OFFSET+1:M,KB+1:N) has been updated.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

JPVT Input and output parameter.

JPVT is INTEGER array, dimension (N)

JPVT(I) = K <==> Column K of the full matrix A has been permuted into position I in AP.

TAU Output parameter.

TAU is COMPLEX*16

TAU is an array, dimension (KB). The scalar factors of the elementary reflectors.

VN1 Input and output parameter.

VN1 is DOUBLE PRECISION

VN1 is an array, dimension (N). The vector with the partial column norms.

VN2 Input and output parameter.

VN2 is DOUBLE PRECISION

VN2 is an array, dimension (N). The vector with the exact column norms.

AUXV Input and output parameter.

AUXV is COMPLEX*16

AUXV is an array, dimension (NB). Auxiliar vector.

F Input and output parameter.

F is COMPLEX*16

F is an array, dimension (LDF, NB). Matrix $F^H = L * Y^H * A$.

LDF Input parameter.

LDF is INTEGER

The leading dimension of the array F. LDF >= max(1, N).

Related Information

For this routine in other precisions, please see *claqps*, *dlaqps* and *slaqps*.

4.17.731 zlaqr0

```
ZLAQR0 computes the eigenvalues of a Hessenberg matrix H
   and, optionally, the matrices T and Z from the Schur decomposition
   H = Z T Z**H, where T is an upper triangular matrix (the
   Schur form), and Z is the unitary matrix of Schur vectors.

   Optionally Z may be postmultiplied into an input unitary
   matrix Q so that this routine can give the Schur factorization
   of a matrix A which has been reduced to the Hessenberg form H
   by the unitary matrix Q:  A = Q*H*Q**H = (QZ)*H*(QZ)**H.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zlaqr0(WANTT, WANTZ, N, ILO, IHI, H, LDH, W, ILOZ, IHIZ, Z, LDZ,
                  WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zlaqr0_(const armpl_int_t *wantt, const armpl_int_t *wantz,
             const armpl_int_t *n, const armpl_int_t *ilo,
             const armpl_int_t *ihi, armpl_doublecomplex_t *h,
             const armpl_int_t *ldh, armpl_doublecomplex_t *w,
             const armpl_int_t *iloz, const armpl_int_t *ihiz,
             armpl_doublecomplex_t *z, const armpl_int_t *ldz,
             armpl_doublecomplex_t *work, const armpl_int_t *lwork,
             armpl_int_t *info);
```

Parameters

WANTT Input parameter.

WANTT is LOGICAL

= .TRUE. : the full Schur form T is required; = .FALSE.: only eigenvalues are required.

WANTZ Input parameter.

WANTZ is LOGICAL

= .TRUE. : the matrix of Schur vectors Z is required; = .FALSE.: Schur vectors are not required.

N Input parameter.

N is INTEGER

The order of the matrix H. N .GE. 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

It is assumed that H is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N and, if ILO.GT.1, H(ILO,ILO-1) is zero. ILO and IHI are normally set by a previous call to ZGEBAL, and then passed to ZGEHRD when the matrix output by ZGEBAL is reduced to Hessenberg form. Otherwise, ILO and IHI should be set to 1 and N, respectively. If N.GT.0, then 1.LE.ILO.LE.IHI.LE.N. If N = 0, then ILO = 1 and IHI = 0.

H Input and output parameter.

H is COMPLEX*16

H is an array, dimension (LDH, N). On entry, the upper Hessenberg matrix H. On exit, if INFO = 0 and WANTT is .TRUE., then H contains the upper triangular matrix T from the Schur decomposition (the Schur form). If INFO = 0 and WANT is .FALSE., then the contents of H are unspecified on exit. (The output value of H when INFO.GT.0 is given under the description of INFO below.)

This subroutine may explicitly set H(i,j) = 0 for i.GT.j and j = 1, 2, ... ILO-1 or j = IHI+1, IHI+2, ... N.

LDH Input parameter.

LDH is INTEGER

The leading dimension of the array H. LDH .GE. max(1, N).

W Output parameter.

W is COMPLEX*16

W is an array, dimension (N). The computed eigenvalues of H(ILO:IHI,ILO:IHI) are stored in W(ILO:IHI). If WANTT is .TRUE., then the eigenvalues are stored in the same order as on the diagonal of the Schur form returned in H, with W(i) = H(i,i).

ILOZ Input parameter.

ILOZ is INTEGER

IHIZ Input parameter.

IHIZ is INTEGER

Specify the rows of Z to which transformations must be applied if WANTZ is .TRUE.. 1 .LE. ILOZ .LE. ILO; IHI .LE. IHIZ .LE. N.

Z Input and output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, IHI). If WANTZ is .FALSE., then Z is not referenced. If WANTZ is .TRUE., then Z(ILO:IHI,ILOZ:IHIZ) is replaced by Z(ILO:IHI,ILOZ:IHIZ)*U where U is the orthogonal Schur factor of H(ILO:IHI,ILO:IHI). (The output value of Z when INFO.GT.0 is given under the description of INFO below.)

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. if WANTZ is .TRUE. then LDZ.GE.MAX(1, IHIZ). Otherwize, LDZ.GE.1.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension LWORK. On exit, if LWORK = -1, WORK(1) returns an estimate of the optimal value for LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK .GE. max(1, N) is sufficient, but LWORK typically as large as 6* N may be required for optimal performance. A workspace query to determine the optimal workspace size is recommended.

If LWORK = -1, then ZLAQR0 does a workspace query. In this case, ZLAQR0 checks the input parameters and estimates the optimal workspace size for the given values of N, ILO and IHI. The estimate is returned in WORK(1). No error message related to LWORK is issued by XERBLA. Neither H nor Z are accessed.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit .GT. 0: if INFO = i, ZLAQR0 failed to compute all of the eigenvalues. Elements 1:ilo-1 and i+1:n of WR and WI contain those eigenvalues which have been successfully computed. (Failures are rare.)

If INFO .GT. 0 and WANT is .FALSE., then on exit, the remaining unconverged eigenvalues are the eigenvalues of the upper Hessenberg matrix rows and columns ILO through INFO of the final, output value of H.

If INFO .GT. 0 and WANTT is .TRUE., then on exit

(*) (initial value of H)*U = U*(final value of H)

where U is a unitary matrix. The final value of H is upper Hessenberg and triangular in rows and columns INFO+1 through IHI.

If INFO .GT. 0 and WANTZ is .TRUE., then on exit

(final value of Z(ILO:IHI,ILOZ:IHIZ) = (initial value of Z(ILO:IHI,ILOZ:IHIZ)*U

where U is the unitary matrix in (*) (regard- less of the value of WANTT.)

If INFO .GT. 0 and WANTZ is .FALSE., then Z is not accessed.

Related Information

For this routine in other precisions, please see *claqr0*, *dlaqr0* and *slaqr0*.

4.17.732 zlaqr1

```
Given a 2-by-2 or 3-by-3 matrix H, ZLAQR1 sets v to a
     scalar multiple of the first column of the product

     (*)  K = (H - s1*I)*(H - s2*I)

     scaling to avoid overflows and most underflows.

     This is useful for starting double implicit shift bulges
     in the QR algorithm.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zlaqr1(N, H, LDH, S1, S2, V)
```

C specification:

```
#include "armpl.h"

void zlaqr1_(const armpl_int_t *n, const armpl_doublecomplex_t *h,
             const armpl_int_t *ldh, const armpl_doublecomplex_t *s1,
             const armpl_doublecomplex_t *s2, armpl_doublecomplex_t *v);
```

Parameters

N Input parameter.

N is INTEGER

Order of the matrix H. N must be either 2 or 3.

H Input parameter.

H is COMPLEX*16

H is an array, dimension (LDH, N). The 2-by-2 or 3-by-3 matrix H in (*).

LDH Input parameter.

LDH is INTEGER

The leading dimension of H as declared in the calling procedure. LDH.GE.N

S1 Input parameter.

S1 is COMPLEX*16

S2 Input parameter.

S2 is COMPLEX*16

S1 and S2 are the shifts defining K in (*) above.

V Output parameter.

V is COMPLEX*16

V is an array, dimension (N). A scalar multiple of the first column of the matrix K in (*).

Related Information

For this routine in other precisions, please see *claqr1*, *dlaqr1* and *slaqr1*.

4.17.733 zlaqr2

```
ZLAQR2 is identical to ZLAQR3 except that it avoids
   recursion by calling ZLAHQR instead of ZLAQR4.

   Aggressive early deflation:

   ZLAQR2 accepts as input an upper Hessenberg matrix
   H and performs an unitary similarity transformation
   designed to detect and deflate fully converged eigenvalues from
   a trailing principal submatrix.  On output H has been over-
   written by a new Hessenberg matrix that is a perturbation of
   an unitary similarity transformation of H.  It is to be
   hoped that the final version of H has many zero subdiagonal
   entries.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zlaqr2(WANTT, WANTZ, N, KTOP, KBOT, NW, H, LDH, ILOZ, IHIZ, Z, LDZ,
                  NS, ND, SH, V, LDV, NH, T, LDT, NV, WV, LDWV, WORK, LWORK)
```

C specification:

```
#include "armpl.h"

void zlaqr2_(const armpl_int_t *wantt, const armpl_int_t *wantz,
             const armpl_int_t *n, const armpl_int_t *ktop,
             const armpl_int_t *kbot, const armpl_int_t *nw,
             armpl_doublecomplex_t *h, const armpl_int_t *ldh,
             const armpl_int_t *iloz, const armpl_int_t *ihiz,
             armpl_doublecomplex_t *z, const armpl_int_t *ldz,
             armpl_int_t *ns, armpl_int_t *nd, armpl_doublecomplex_t *sh,
             armpl_doublecomplex_t *v, const armpl_int_t *ldv,
             const armpl_int_t *nh, armpl_doublecomplex_t *t,
             const armpl_int_t *ldt, const armpl_int_t *nv,
             armpl_doublecomplex_t *wv, const armpl_int_t *ldwv,
             armpl_doublecomplex_t *work, const armpl_int_t *lwork);
```

Parameters

WANTT Input parameter.

WANTT is LOGICAL

If .TRUE., then the Hessenberg matrix H is fully updated so that the triangular Schur factor may be computed (in cooperation with the calling subroutine). If .FALSE., then only enough of H is updated to preserve the eigenvalues.

WANTZ Input parameter.

WANTZ is LOGICAL

If .TRUE., then the unitary matrix Z is updated so so that the unitary Schur factor may be computed (in cooperation with the calling subroutine). If .FALSE., then Z is not referenced.

N Input parameter.

N is INTEGER

The order of the matrix H and (if WANTZ is .TRUE.) the order of the unitary matrix Z.

KTOP Input parameter.

KTOP is INTEGER

It is assumed that either KTOP = 1 or H(KTOP,KTOP-1)=0. KBOT and KTOP together determine an isolated block along the diagonal of the Hessenberg matrix.

KBOT Input parameter.

KBOT is INTEGER

It is assumed without a check that either KBOT = N or H(KBOT+1, KBOT)=0. KBOT and KTOP together determine an isolated block along the diagonal of the Hessenberg matrix.

NW Input parameter.

NW is INTEGER

Deflation window size. 1 .LE. NW .LE. (KBOT-KTOP+1).

H Input and output parameter.

H is COMPLEX*16

H is an array, dimension (LDH, N). On input the initial N-by-N section of H stores the Hessenberg matrix undergoing aggressive early deflation. On output H has been transformed by a unitary similarity transformation, perturbed, and the returned to Hessenberg form that (it is to be hoped) has some zero subdiagonal entries.

LDH Input parameter.

LDH is INTEGER

Leading dimension of H just as declared in the calling subroutine. N .LE. LDH

ILOZ Input parameter.

ILOZ is INTEGER

IHIZ Input parameter.

IHIZ is INTEGER

Specify the rows of Z to which transformations must be applied if WANTZ is .TRUE.. 1 .LE. ILOZ .LE. IHIZ .LE. N.

Z Input and output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, N). IF WANTZ is .TRUE., then on output, the unitary similarity transformation mentioned above has been accumulated into Z(ILOZ:IHIZ,ILOZ:IHIZ) from the right. If WANTZ is .FALSE., then Z is unreferenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of Z just as declared in the calling subroutine. 1 .LE. LDZ.

NS Output parameter.

NS is INTEGER

The number of unconverged (ie approximate) eigenvalues returned in SR and SI that may be used as shifts by the calling subroutine.

ND Output parameter.

ND is INTEGER

The number of converged eigenvalues uncovered by this subroutine.

SH Output parameter.

SH is COMPLEX*16

SH is an array, dimension (KBOT). On output, approximate eigenvalues that may be used for shifts are stored in SH(KBOT-ND-NS+1) through SR(KBOT-ND). Converged eigenvalues are stored in SH(KBOT-ND+1) through SH(KBOT).

V Output parameter.

V is COMPLEX*16

V is an array, dimension (LDV, NW). An NW-by-NW work array.

LDV Input parameter.

LDV is INTEGER

The leading dimension of V just as declared in the calling subroutine. NW .LE. LDV

NH Input parameter.

NH is INTEGER

The number of columns of T. NH.GE.NW.

T Output parameter.

T is COMPLEX*16

T is an array, dimension (LDT, NW) .

LDT Input parameter.

LDT is INTEGER

The leading dimension of T just as declared in the calling subroutine. NW .LE. LDT

NV Input parameter.

NV is INTEGER

The number of rows of work array WV available for workspace. NV.GE.NW.

WV Output parameter.

WV is COMPLEX*16

WV is an array, dimension (LDWV, NW) .

LDWV Input parameter.

LDWV is INTEGER

The leading dimension of W just as declared in the calling subroutine. NW .LE. LDV

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (LWORK). On exit, WORK(1) is set to an estimate of the optimal value of LWORK for the given values of N, NW, KTOP and KBOT.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the work array WORK. LWORK = 2*NW suffices, but greater efficiency may result from larger values of LWORK.

If LWORK = -1, then a workspace query is assumed; ZLAQR2 only estimates the optimal workspace size for the given values of N, NW, KTOP and KBOT. The estimate is returned in WORK(1). No error message related to LWORK is issued by XERBLA. Neither H nor Z are accessed.

Related Information

For this routine in other precisions, please see *claqr2*, *dlaqr2* and *slaqr2*.

4.17.734 zlaqr3

```
Aggressive early deflation:

   ZLAQR3 accepts as input an upper Hessenberg matrix
   H and performs an unitary similarity transformation
   designed to detect and deflate fully converged eigenvalues from
   a trailing principal submatrix.  On output H has been over-
   written by a new Hessenberg matrix that is a perturbation of
   an unitary similarity transformation of H.  It is to be
   hoped that the final version of H has many zero subdiagonal
   entries.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zlaqr3(WANTT, WANTZ, N, KTOP, KBOT, NW, H, LDH, ILOZ, IHIZ, Z, LDZ,
                  NS, ND, SH, V, LDV, NH, T, LDT, NV, WV, LDWV, WORK, LWORK)
```

C specification:

```
#include "armpl.h"

void zlaqr3_(const armpl_int_t *wantt, const armpl_int_t *wantz,
             const armpl_int_t *n, const armpl_int_t *ktop,
             const armpl_int_t *kbot, const armpl_int_t *nw,
             armpl_doublecomplex_t *h, const armpl_int_t *ldh,
             const armpl_int_t *iloz, const armpl_int_t *ihiz,
             armpl_doublecomplex_t *z, const armpl_int_t *ldz,
             armpl_int_t *ns, armpl_int_t *nd, armpl_doublecomplex_t *sh,
             armpl_doublecomplex_t *v, const armpl_int_t *ldv,
             const armpl_int_t *nh, armpl_doublecomplex_t *t,
             const armpl_int_t *ldt, const armpl_int_t *nv,
             armpl_doublecomplex_t *wv, const armpl_int_t *ldwv,
             armpl_doublecomplex_t *work, const armpl_int_t *lwork);
```

Parameters

WANTT Input parameter.

WANTT is LOGICAL

If .TRUE., then the Hessenberg matrix H is fully updated so that the triangular Schur factor may be computed (in cooperation with the calling subroutine). If .FALSE., then only enough of H is updated to preserve the eigenvalues.

WANTZ Input parameter.

WANTZ is LOGICAL

If .TRUE., then the unitary matrix Z is updated so so that the unitary Schur factor may be computed (in cooperation with the calling subroutine). If .FALSE., then Z is not referenced.

N Input parameter.

N is INTEGER

The order of the matrix H and (if WANTZ is .TRUE.) the order of the unitary matrix Z.

KTOP Input parameter.

KTOP is INTEGER

It is assumed that either KTOP = 1 or H(KTOP,KTOP-1)=0. KBOT and KTOP together determine an isolated block along the diagonal of the Hessenberg matrix.

KBOT Input parameter.

KBOT is INTEGER

It is assumed without a check that either KBOT = N or H(KBOT+1, KBOT)=0. KBOT and KTOP together determine an isolated block along the diagonal of the Hessenberg matrix.

NW Input parameter.

NW is INTEGER

Deflation window size. 1 .LE. NW .LE. (KBOT-KTOP+1).

H Input and output parameter.

H is COMPLEX*16

H is an array, dimension (LDH, N). On input the initial N-by-N section of H stores the Hessenberg matrix undergoing aggressive early deflation. On output H has been transformed by a unitary similarity transformation, perturbed, and the returned to Hessenberg form that (it is to be hoped) has some zero subdiagonal entries.

LDH Input parameter.

LDH is INTEGER

Leading dimension of H just as declared in the calling subroutine. N .LE. LDH

ILOZ Input parameter.

ILOZ is INTEGER

IHIZ Input parameter.

IHIZ is INTEGER

Specify the rows of Z to which transformations must be applied if WANTZ is .TRUE.. 1 .LE. ILOZ .LE. IHIZ .LE. N.

Z Input and output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, N). IF WANTZ is .TRUE., then on output, the unitary similarity transformation mentioned above has been accumulated into Z(ILOZ:IHIZ,ILOZ:IHIZ) from the right. If WANTZ is .FALSE., then Z is unreferenced.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of Z just as declared in the calling subroutine. 1 .LE. LDZ.

NS Output parameter.

NS is INTEGER

The number of unconverged (ie approximate) eigenvalues returned in SR and SI that may be used as shifts by the calling subroutine.

ND Output parameter.

ND is INTEGER

The number of converged eigenvalues uncovered by this subroutine.

SH Output parameter.

SH is COMPLEX*16

SH is an array, dimension (KBOT). On output, approximate eigenvalues that may be used for shifts are stored in SH(KBOT-ND-NS+1) through SR(KBOT-ND). Converged eigenvalues are stored in SH(KBOT-ND+1) through SH(KBOT).

V Output parameter.

V is COMPLEX*16

V is an array, dimension (LDV, NW). An NW-by-NW work array.

LDV Input parameter.

LDV is INTEGER

The leading dimension of V just as declared in the calling subroutine. NW .LE. LDV

NH Input parameter.

NH is INTEGER

The number of columns of T. NH.GE.NW.

T Output parameter.

T is COMPLEX*16

T is an array, dimension (LDT, NW) .

LDT Input parameter.

LDT is INTEGER

The leading dimension of T just as declared in the calling subroutine. NW .LE. LDT

NV Input parameter.

NV is INTEGER

The number of rows of work array WV available for workspace. NV.GE.NW.

WV Output parameter.

WV is COMPLEX*16

WV is an array, dimension (LDWV, NW) .

LDWV Input parameter.

LDWV is INTEGER

The leading dimension of W just as declared in the calling subroutine. NW .LE. LDV

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (LWORK). On exit, WORK(1) is set to an estimate of the optimal value of LWORK for the given values of N, NW, KTOP and KBOT.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the work array WORK. LWORK = 2*NW suffices, but greater efficiency may result from larger values of LWORK.

If LWORK = -1, then a workspace query is assumed; ZLAQR3 only estimates the optimal workspace size for the given values of N, NW, KTOP and KBOT. The estimate is returned in WORK(1). No error message related to LWORK is issued by XERBLA. Neither H nor Z are accessed.

Related Information

For this routine in other precisions, please see *claqr3*, *dlaqr3* and *slaqr3*.

4.17.735 zlaqr4

```
ZLAQR4 implements one level of recursion for ZLAQR0.
   It is a complete implementation of the small bulge multi-shift
   QR algorithm.  It may be called by ZLAQR0 and, for large enough
   deflation window size, it may be called by ZLAQR3.  This
   subroutine is identical to ZLAQR0 except that it calls ZLAQR2
   instead of ZLAQR3.

   ZLAQR4 computes the eigenvalues of a Hessenberg matrix H
   and, optionally, the matrices T and Z from the Schur decomposition
   H = Z T Z**H, where T is an upper triangular matrix (the
   Schur form), and Z is the unitary matrix of Schur vectors.

   Optionally Z may be postmultiplied into an input unitary
   matrix Q so that this routine can give the Schur factorization
   of a matrix A which has been reduced to the Hessenberg form H
   by the unitary matrix Q:  A = Q*H*Q**H = (QZ)*H*(QZ)**H.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zlaqr4(WANTT, WANTZ, N, ILO, IHI, H, LDH, W, ILOZ, IHIZ, Z, LDZ,
                  WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zlaqr4_(const armpl_int_t *wantt, const armpl_int_t *wantz,
             const armpl_int_t *n, const armpl_int_t *ilo,
             const armpl_int_t *ihi, armpl_doublecomplex_t *h,
             const armpl_int_t *ldh, armpl_doublecomplex_t *w,
             armpl_int_t *iloz, armpl_int_t *ihiz, armpl_doublecomplex_t *z,
             const armpl_int_t *ldz, armpl_doublecomplex_t *work,
             const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

WANTT Input parameter.

WANTT is LOGICAL

= .TRUE. : the full Schur form T is required; = .FALSE.: only eigenvalues are required.

WANTZ Input parameter.

WANTZ is LOGICAL

= .TRUE. : the matrix of Schur vectors Z is required; = .FALSE.: Schur vectors are not required.

N Input parameter.

N is INTEGER

The order of the matrix H. N .GE. 0.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

It is assumed that H is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N and, if ILO.GT.1, H(ILO,ILO-1) is zero. ILO and IHI are normally set by a previous call to ZGEBAL, and then passed to ZGEHRD when the matrix output by ZGEBAL is reduced to Hessenberg form. Otherwise, ILO and IHI should be set to 1 and N, respectively. If N.GT.0, then 1.LE.ILO.LE.IHI.LE.N. If N = 0, then ILO = 1 and IHI = 0.

H Input and output parameter.

H is COMPLEX*16

H is an array, dimension (LDH, N). On entry, the upper Hessenberg matrix H. On exit, if INFO = 0 and WANTT is .TRUE., then H contains the upper triangular matrix T from the Schur decomposition (the Schur form). If INFO = 0 and WANT is .FALSE., then the contents of H are unspecified on exit. (The output value of H when INFO.GT.0 is given under the description of INFO below.)

This subroutine may explicitly set H(i,j) = 0 for i.GT.j and j = 1, 2, ... ILO-1 or j = IHI+1, IHI+2, ... N.

LDH Input parameter.

LDH is INTEGER

The leading dimension of the array H. LDH .GE. max(1, N).

W Output parameter.

W is COMPLEX*16

W is an array, dimension (N). The computed eigenvalues of H(ILO:IHI,ILO:IHI) are stored in W(ILO:IHI). If WANTT is .TRUE., then the eigenvalues are stored in the same order as on the diagonal of the Schur form returned in H, with W(i) = H(i,i).

ILOZ Input parameter.

ILOZ is INTEGER

IHIZ Input parameter.

IHIZ is INTEGER

Specify the rows of Z to which transformations must be applied if WANTZ is .TRUE.. 1 .LE. ILOZ .LE. ILO; IHI .LE. IHIZ .LE. N.

Z Input and output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, IHI). If WANTZ is .FALSE., then Z is not referenced. If WANTZ is .TRUE., then Z(ILO:IHI,ILOZ:IHIZ) is replaced by Z(ILO:IHI,ILOZ:IHIZ)*U where U is the orthogonal Schur factor

of H(ILO:IHI,ILO:IHI). (The output value of Z when INFO.GT.0 is given under the description of INFO below.)

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. if WANTZ is .TRUE. then LDZ.GE.MAX(1, IHIZ). Otherwize, LDZ.GE.1.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension LWORK. On exit, if LWORK = -1, WORK(1) returns an estimate of the optimal value for LWORK.

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK .GE. max(1, N) is sufficient, but LWORK typically as large as 6* N may be required for optimal performance. A workspace query to determine the optimal workspace size is recommended.

If LWORK = -1, then ZLAQR4 does a workspace query. In this case, ZLAQR4 checks the input parameters and estimates the optimal workspace size for the given values of N, ILO and IHI. The estimate is returned in WORK(1). No error message related to LWORK is issued by XERBLA. Neither H nor Z are accessed.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit .GT. 0: if INFO = i, ZLAQR4 failed to compute all of the eigenvalues. Elements 1:ilo-1 and i+1:n of WR and WI contain those eigenvalues which have been successfully computed. (Failures are rare.)

If INFO .GT. 0 and WANT is .FALSE., then on exit, the remaining unconverged eigenvalues are the eigenvalues of the upper Hessenberg matrix rows and columns ILO through INFO of the final, output value of H.

If INFO .GT. 0 and WANTT is .TRUE., then on exit

(*) (initial value of H)*U = U*(final value of H)

where U is a unitary matrix. The final value of H is upper Hessenberg and triangular in rows and columns INFO+1 through IHI.

If INFO .GT. 0 and WANTZ is .TRUE., then on exit

(final value of Z(ILO:IHI,ILOZ:IHIZ) = (initial value of Z(ILO:IHI,ILOZ:IHIZ)*U

where U is the unitary matrix in (*) (regard- less of the value of WANTT.)

If INFO .GT. 0 and WANTZ is .FALSE., then Z is not accessed.

Related Information

For this routine in other precisions, please see *claqr4*, *dlaqr4* and *slaqr4*.

4.17.736 zlaqr5

```
ZLAQR5, called by ZLAQR0, performs a
   single small-bulge multi-shift QR sweep.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zlaqr5(WANTT, WANTZ, KACC22, N, KTOP, KBOT, NSHFTS, S, H, LDH,
                  ILOZ, IHIZ, Z, LDZ, V, LDV, U, LDU, NV, WV, LDWV, NH, WH,
                  LDWH)
```

C specification:

```
#include "armpl.h"

void zlaqr5_(const armpl_int_t *wantt, const armpl_int_t *wantz,
             const armpl_int_t *kacc22, const armpl_int_t *n,
             const armpl_int_t *ktop, const armpl_int_t *kbot,
             const armpl_int_t *nshfts, armpl_doublecomplex_t *s,
             armpl_doublecomplex_t *h, const armpl_int_t *ldh,
             const armpl_int_t *iloz, const armpl_int_t *ihiz,
             armpl_doublecomplex_t *z, const armpl_int_t *ldz,
             armpl_doublecomplex_t *v, const armpl_int_t *ldv,
             armpl_doublecomplex_t *u, const armpl_int_t *ldu,
             const armpl_int_t *nv, armpl_doublecomplex_t *wv,
             const armpl_int_t *ldwv, const armpl_int_t *nh,
             armpl_doublecomplex_t *wh, const armpl_int_t *ldwh);
```

Parameters

WANTT Input parameter.

WANTT is LOGICAL

WANTT = .true. if the triangular Schur factor is being computed. WANTT is set to .false. otherwise.

WANTZ Input parameter.

WANTZ is LOGICAL

WANTZ = .true. if the unitary Schur factor is being computed. WANTZ is set to .false. otherwise.

KACC22 Input parameter.

KACC22 is INTEGER with value 0, 1, or 2.

Specifies the computation mode of far-from-diagonal orthogonal updates. = 0: ZLAQR5 does not accumulate reflections and does not use matrix-matrix multiply to update far-from-diagonal matrix entries. = 1: ZLAQR5 accumulates reflections and uses matrix-matrix multiply to update the far-from-diagonal matrix entries. = 2: ZLAQR5 accumulates reflections, uses matrix-matrix multiply to update the far-from-diagonal matrix entries, and takes advantage of 2-by-2 block structure during matrix multiplies.

N Input parameter.

N is INTEGER

N is the order of the Hessenberg matrix H upon which this subroutine operates.

KTOP Input parameter.

KTOP is INTEGER

KBOT Input parameter.

KBOT is INTEGER

These are the first and last rows and columns of an isolated diagonal block upon which the QR sweep is to be applied. It is assumed without a check that either KTOP = 1 or H(KTOP,KTOP-1) = 0 and either KBOT = N or H(KBOT+1, KBOT) = 0.

NSHFTS Input parameter.

NSHFTS is INTEGER

NSHFTS gives the number of simultaneous shifts. NSHFTS must be positive and even.

S Input and output parameter.

S is COMPLEX*16

S is an array, dimension (NSHFTS). S contains the shifts of origin that define the multi- shift QR sweep. On output S may be reordered.

H Input and output parameter.

H is COMPLEX*16

H is an array, dimension (LDH, N). On input H contains a Hessenberg matrix. On output a multi-shift QR sweep with shifts SR(J)+i*SI(J) is applied to the isolated diagonal block in rows and columns KTOP through KBOT.

LDH Input parameter.

LDH is INTEGER

LDH is the leading dimension of H just as declared in the calling procedure. LDH.GE.MAX(1, N).

ILOZ Input parameter.

ILOZ is INTEGER

IHIZ Input parameter.

IHIZ is INTEGER

Specify the rows of Z to which transformations must be applied if WANTZ is .TRUE.. 1 .LE. ILOZ .LE. IHIZ .LE. N

Z Input and output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, IHIZ). If WANTZ = .TRUE., then the QR Sweep unitary similarity transformation is accumulated into Z(ILOZ:IHIZ,ILOZ:IHIZ) from the right. If WANTZ = .FALSE., then Z is unreferenced.

LDZ Input parameter.

LDZ is INTEGER

LDA is the leading dimension of Z just as declared in the calling procedure. LDZ.GE.N.

V Output parameter.

V is COMPLEX*16

V is an array, dimension (LDV,NSHFTS/2) .

LDV Input parameter.

LDV is INTEGER

LDV is the leading dimension of V as declared in the calling procedure. LDV.GE.3.

U Output parameter.

U is COMPLEX*16

U is an array, dimension (LDU,3*NSHFTS-3) .

LDU Input parameter.

LDU is INTEGER

LDU is the leading dimension of U just as declared in the in the calling subroutine. LDU.GE.3*NSHFTS-3.

NH Input parameter.

NH is INTEGER

NH is the number of columns in array WH available for workspace. NH.GE.1.

WH Output parameter.

WH is COMPLEX*16

WH is an array, dimension (LDWH, NH) .

LDWH Input parameter.

LDWH is INTEGER

Leading dimension of WH just as declared in the calling procedure. LDWH.GE.3*NSHFTS-3.

NV Input parameter.

NV is INTEGER

NV is the number of rows in WV agailable for workspace. NV.GE.1.

WV Output parameter.

WV is COMPLEX*16

WV is an array, dimension (LDWV,3*NSHFTS-3) .

LDWV Input parameter.

LDWV is INTEGER

LDWV is the leading dimension of WV as declared in the in the calling subroutine. LDWV.GE.NV.

Related Information

For this routine in other precisions, please see *claqr5*, *dlaqr5* and *slaqr5*.

4.17.737 zlaqsb

zlaqsb equilibrates a symmetric band matrix A using the scaling factors in the vector S.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlaqsb(UPLO, N, KD, AB, LDAB, S, SCOND, AMAX, EQUED)
```

C specification:

```
#include "armpl.h"

void zlaqsb_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
             armpl_doublecomplex_t *ab, const armpl_int_t *ldab,
             const double *s, const double *scond, const double *amax,
             char *equed, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored. = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of super-diagonals of the matrix A if UPLO = 'U', or the number of sub-diagonals if UPLO = 'L'. KD >= 0.

AB Input and output parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, if INFO = 0, the triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H of the band matrix A, in the same storage format as A.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

S Input parameter.

S is DOUBLE PRECISION

S is an array, dimension (N). The scale factors for A.

SCOND Input parameter.

SCOND is DOUBLE PRECISION

Ratio of the smallest S(i) to the largest S(i).

AMAX Input parameter.

AMAX is DOUBLE PRECISION

Absolute value of largest matrix entry.

EQUED Output parameter.

EQUED is CHARACTER*1

Specifies whether or not equilibration was done. = 'N': No equilibration. = 'Y': Equilibration was done, i.e., A has been replaced by diag(S) * A * diag(S).

Related Information

For this routine in other precisions, please see *claqsb*, *dlaqsb* and *slaqsb*.

4.17.738 zlaqsp

`zlaqsp` equilibrates a symmetric matrix A using the scaling factors in the vector S.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlaqsp(UPLO, N, AP, S, SCOND, AMAX, EQUED)
```

C specification:

```
#include "armpl.h"

void zlaqsp_(const char *uplo, const armpl_int_t *n,
             armpl_doublecomplex_t *ap, const double *s, const double *scond,
             const double *amax, char *equed, ... );
```

Parameters

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored. = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

`N` is `INTEGER`

The order of the matrix A. `N` >= 0.

AP Input and output parameter.

`AP` is `COMPLEX*16`

`AP` is an array, dimension (N*(N+1)/2). On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array `AP` as follows: if `UPLO` = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if `UPLO` = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

On exit, the equilibrated matrix: diag(`S`) * A * diag(`S`), in the same storage format as A.

S Input parameter.

`S` is `DOUBLE PRECISION`

`S` is an array, dimension (`N`). The scale factors for A.

SCOND Input parameter.

`SCOND` is `DOUBLE PRECISION`

Ratio of the smallest S(i) to the largest S(i).

AMAX Input parameter.

`AMAX` is `DOUBLE PRECISION`

Absolute value of largest matrix entry.

EQUED Output parameter.

`EQUED` is `CHARACTER*1`

Specifies whether or not equilibration was done. = 'N': No equilibration. = 'Y': Equilibration was done, i.e., A has been replaced by diag(S) * A * diag(S).

Related Information

For this routine in other precisions, please see *claqsp*, *dlaqsp* and *slaqsp*.

4.17.739 zlaqsy

zlaqsy equilibrates a symmetric matrix A using the scaling factors in the vector S.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlaqsy(UPLO, N, A, LDA, S, SCOND, AMAX, EQUED)
```

C specification:

```
#include "armpl.h"

void zlaqsy_(const char *uplo, const armpl_int_t *n, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, const double *s, const double *scond,
             const double *amax, char *equed, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored. = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading n by n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n by n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if EQUED = 'Y', the equilibrated matrix: diag(S) * A * diag(S).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(N,1).

S Input parameter.

S is DOUBLE PRECISION

S is an array, dimension (N). The scale factors for A.

SCOND Input parameter.

SCOND is DOUBLE PRECISION

Ratio of the smallest S(i) to the largest S(i).

AMAX Input parameter.

AMAX is DOUBLE PRECISION

Absolute value of largest matrix entry.

EQUED Output parameter.

EQUED is CHARACTER*1

Specifies whether or not equilibration was done. = 'N': No equilibration. = 'Y': Equilibration was done, i.e., A has been replaced by diag(S) * A * diag(S).

Related Information

For this routine in other precisions, please see *claqsy*, *dlaqsy* and *slaqsy*.

4.17.740 zlar1v

zlar1v computes the (scaled) r-th column of the inverse of the sumbmatrix in rows B1 through BN of the tridiagonal matrix L D L^T - sigma I. When sigma is close to an eigenvalue, the computed vector is an accurate eigenvector. Usually, r corresponds to the index where the eigenvector is largest in magnitude. The following steps accomplish this computation : (a) Stationary qd transform, L D L^T - sigma I = L(+) D(+) $L(+)^T$, (b) Progressive qd transform, L D L^T - sigma I = U(-) D(-) $U(-)^T$, (c) Computation of the diagonal elements of the inverse of

```
L D L**T - sigma I by combining the above transforms, and choosing
r as the index where the diagonal of the inverse is (one of the)
largest in magnitude.
```

(d) Computation of the (scaled) r-th column of the inverse using the

```
twisted factorization obtained by combining the top part of the
the stationary and the bottom part of the progressive transform.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zlar1v(N, B1, BN, LAMBDA, D, L, LD, LLD, PIVMIN, GAPTOL, Z, WANTNC,
                  NEGCNT, ZTZ, MINGMA, R, ISUPPZ, NRMINV, RESID, RQCORR,
                  WORK)
```

C specification:

```
#include "armpl.h"

void zlar1v_(const armpl_int_t *n, const armpl_int_t *b1,
             const armpl_int_t *bn, const double *lambda, const double *d,
```

(continues on next page)

(continued from previous page)

```
                const double *l, const double *ld, const double *lld,
                const double *pivmin, const double *gaptol,
                armpl_doublecomplex_t *z, const armpl_int_t *wantnc,
                armpl_int_t *negcnt, double *ztz, double *mingma, armpl_int_t *r,
                armpl_int_t *isuppz, double *nrminv, double *resid,
                double *rqcorr, double *work);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix L D L^T .

B1 Input parameter.

B1 is INTEGER

First index of the submatrix of L D L^T .

BN Input parameter.

BN is INTEGER

Last index of the submatrix of L D L^T .

LAMBDA Input parameter.

LAMBDA is DOUBLE PRECISION

The shift. In order to compute an accurate eigenvector, LAMBDA should be a good approximation to an eigenvalue of L D L^T .

L Input parameter.

L is DOUBLE PRECISION

L is an array, dimension (N-1). The (n-1) subdiagonal elements of the unit bidiagonal matrix L, in elements 1 to N-1.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The n diagonal elements of the diagonal matrix D.

LD Input parameter.

LD is DOUBLE PRECISION

LD is an array, dimension (N-1). The n-1 elements L(i)*D(i).

LLD Input parameter.

LLD is DOUBLE PRECISION

LLD is an array, dimension (N-1). The n-1 elements L(i)*L(i)*D(i).

PIVMIN Input parameter.

PIVMIN is DOUBLE PRECISION

The minimum pivot in the Sturm sequence.

GAPTOL Input parameter.

GAPTOL is DOUBLE PRECISION

Tolerance that indicates when eigenvector entries are negligible w.r.t. their contribution to the residual.

Z Input and output parameter.

Z is COMPLEX*16

Z is an array, dimension (N). On input, all entries of Z must be set to 0. On output, Z contains the (scaled) r-th column of the inverse. The scaling is such that Z(R) equals 1.

WANTNC Input parameter.

WANTNC is LOGICAL

Specifies whether NEGCNT has to be computed.

NEGCNT Output parameter.

NEGCNT is INTEGER

If WANTNC is .TRUE. then NEGCNT = the number of pivots < pivmin in the matrix factorization L D L^T , and NEGCNT = -1 otherwise.

ZTZ Output parameter.

ZTZ is DOUBLE PRECISION

The square of the 2-norm of Z.

MINGMA Output parameter.

MINGMA is DOUBLE PRECISION

The reciprocal of the largest (in magnitude) diagonal element of the inverse of L D L^T - sigma I.

R Input and output parameter.

R is INTEGER

The twist index for the twisted factorization used to compute Z. On input, 0 <= R <= N. If R is input as 0, R is set to the index where (L D L^T - sigma I)^{-1} is largest in magnitude. If 1 <= R <= N, R is unchanged. On output, R contains the twist index used to compute Z. Ideally, R designates the position of the maximum entry in the eigenvector.

ISUPPZ Output parameter.

ISUPPZ is INTEGER array, dimension (2)

The support of the vector in Z, i.e., the vector Z is nonzero only in elements ISUPPZ(1) through ISUPPZ(2).

NRMINV Output parameter.

NRMINV is DOUBLE PRECISION

NRMINV = 1/SQRT(ZTZ)

RESID Output parameter.

RESID is DOUBLE PRECISION

The residual of the FP vector. RESID = ABS(MINGMA)/SQRT(ZTZ)

RQCORR Output parameter.

RQCORR is DOUBLE PRECISION

The Rayleigh Quotient correction to LAMBDA. RQCORR = MINGMA*TMP

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (4*N) .

Related Information

For this routine in other precisions, please see *clar1v*, *dlar1v* and *slar1v*.

4.17.741 zlar2v

`zlar2v` applies a vector of complex plane rotations with real cosines from both sides to a sequence of 2-by-2 complex Hermitian matrices, defined by the elements of the vectors x, y and z. For i = 1,2,...,n

```
(       x(i)  z(i) ) :=
( conjg(z(i)) y(i) )

  (  c(i) conjg(s(i)) ) (       x(i)  z(i) ) ( c(i) -conjg(s(i)) )
  ( -s(i)       c(i)  ) ( conjg(z(i)) y(i) ) ( s(i)        c(i)  )
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zlar2v(N, X, Y, Z, INCX, C, S, INCC)
```

C specification:

```
#include "armpl.h"

void zlar2v_(const armpl_int_t *n, armpl_doublecomplex_t *x,
             armpl_doublecomplex_t *y, armpl_doublecomplex_t *z,
             const armpl_int_t *incx, const double *c,
             const armpl_doublecomplex_t *s, const armpl_int_t *incc);
```

Parameters

N Input parameter.

`N` is `INTEGER`

The number of plane rotations to be applied.

X Input and output parameter.

`X` is `COMPLEX*16`

`X` is an array, dimension (1+(N-1)*INCX). The vector x; the elements of x are assumed to be real.

Y Input and output parameter.

`Y` is `COMPLEX*16`

`Y` is an array, dimension (1+(N-1)*INCX). The vector y; the elements of y are assumed to be real.

Z Input and output parameter.

`Z` is `COMPLEX*16`

`Z` is an array, dimension (1+(N-1)*INCX). The vector z.

INCX Input parameter.

`INCX` is `INTEGER`

The increment between elements of `X`, `Y` and `Z`. `INCX` > 0.

C Input parameter.

C is DOUBLE PRECISION

C is an array, dimension (1+(N-1)*INCC). The cosines of the plane rotations.

S Input parameter.

S is COMPLEX*16

S is an array, dimension (1+(N-1)*INCC). The sines of the plane rotations.

INCC Input parameter.

INCC is INTEGER

The increment between elements of C and S. INCC > 0.

Related Information

For this routine in other precisions, please see *clar2v*, *dlar2v* and *slar2v*.

4.17.742 zlarcm

zlarcm performs a very simple matrix-matrix multiplication:

```
C := A * B,
```

where A is M by M and real; B is M by N and complex; C is M by N and complex.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlarcm(M, N, A, LDA, B, LDB, C, LDC, RWORK)
```

C specification:

```
#include "armpl.h"

void zlarcm_(const armpl_int_t *m, const armpl_int_t *n, const double *a,
             const armpl_int_t *lda, const armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, armpl_doublecomplex_t *c,
             const armpl_int_t *ldc, double *rwork);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A and of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns and rows of the matrix B and the number of columns of the matrix C. N >= 0.

A Input parameter.

A is DOUBLE PRECISION

A is an array, dimension (LDA, M). On entry, A contains the M by M matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >=max(1, M).

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). On entry, B contains the M by N matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >=max(1, M).

C Output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). On exit, C contains the M by N matrix C.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >=max(1, M).

RWORK Output parameter.

RWORK is DOUBLE PRECISION

RWORK is an array, dimension (2*M*N) .

Related Information

For this routine in other precisions, please see *clarcm*. It also exists with a native C interface as *LAPACKE_zlarcm*.

4.17.743 zlarf

zlarf applies a complex elementary reflector H to a complex M-by-N matrix C, from either the left or the right. H is represented in the form

```
H = I - tau * v * v**H
```

where tau is a complex scalar and v is a complex vector.

If tau = 0, then H is taken to be the unit matrix.

To apply H^H , supply conjg(tau) instead tau.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlarf(SIDE, M, N, V, INCV, TAU, C, LDC, WORK)
```

C specification:

```
#include "armpl.h"

void zlarf_(const char *side, const armpl_int_t *m, const armpl_int_t *n,
            const armpl_doublecomplex_t *v, const armpl_int_t *incv,
            const armpl_doublecomplex_t *tau, armpl_doublecomplex_t *c,
            const armpl_int_t *ldc, armpl_doublecomplex_t *work, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': form H * C = 'R': form C * H

M Input parameter.

M is INTEGER

The number of rows of the matrix C.

N Input parameter.

N is INTEGER

The number of columns of the matrix C.

V Input parameter.

V is COMPLEX*16

V is an array, dimension. (1 + (M-1)*abs(INCV)) if SIDE = 'L' or (1 + (N-1)*abs(INCV)) if SIDE = 'R' The vector v in the representation of H. V is not used if TAU = 0.

INCV Input parameter.

INCV is INTEGER

The increment between elements of v. INCV <> 0.

TAU Input parameter.

TAU is COMPLEX*16

The value tau in the representation of H.

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by the matrix H * C if SIDE = 'L', or C * H if SIDE = 'R'.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension. (N) if SIDE = 'L' or (M) if SIDE = 'R'

Related Information

For this routine in other precisions, please see *clarf*, *dlarf* and *slarf*.

4.17.744 zlarfb

`zlarfb` applies a complex block reflector H or its transpose H^H to a complex M-by-N matrix C, from either the left or the right.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlarfb(SIDE, TRANS, DIRECT, STOREV, M, N, K, V, LDV, T, LDT, C,
                  LDC, WORK, LDWORK)
```

C specification:

```
#include "armpl.h"

void zlarfb_(const char *side, const char *trans, const char *direct,
             const char *storev, const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *k, const armpl_doublecomplex_t *v,
             const armpl_int_t *ldv, const armpl_doublecomplex_t *t,
             const armpl_int_t *ldt, armpl_doublecomplex_t *c,
             const armpl_int_t *ldc, armpl_doublecomplex_t *work,
             const armpl_int_t *ldwork, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply H or H^H from the Left = 'R': apply H or H^H from the Right

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply H (No transpose) = 'C': apply H^H (Conjugate transpose)

DIRECT Input parameter.

DIRECT is CHARACTER*1

Indicates how H is formed from a product of elementary reflectors = 'F': H = H(1) H(2) . . . H(k) (Forward) = 'B': H = H(k) . . . H(2) H(1) (Backward)

STOREV Input parameter.

STOREV is CHARACTER*1

Indicates how the vectors which define the elementary reflectors are stored: = 'C': Columnwise = 'R': Rowwise

M Input parameter.

M is INTEGER

The number of rows of the matrix C.

N Input parameter.

N is INTEGER

The number of columns of the matrix C.

K Input parameter.

K is INTEGER

The order of the matrix T (= the number of elementary reflectors whose product defines the block reflector).

V Input parameter.

V is COMPLEX*16

V is an array, dimension. (LDV, K) if STOREV = 'C' (LDV, M) if STOREV = 'R' and SIDE = 'L' (LDV, N) if STOREV = 'R' and SIDE = 'R' See Further Details.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. If STOREV = 'C' and SIDE = 'L', LDV >= max(1, M); if STOREV = 'C' and SIDE = 'R', LDV >= max(1, N); if STOREV = 'R', LDV >= K.

T Input parameter.

T is COMPLEX*16

T is an array, dimension (LDT, K). The triangular K-by-K matrix T in the representation of the block reflector.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= K.

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by H* C or H^H * C or C*H or C*H^H .

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (LDWORK, K) .

LDWORK Input parameter.

LDWORK is INTEGER

The leading dimension of the array WORK. If SIDE = 'L', LDWORK >= max(1, N); if SIDE = 'R', LDWORK >= max(1, M).

Related Information

For this routine in other precisions, please see *clarfb*, *dlarfb* and *slarfb*. It also exists with a native C interface as *LAPACKE_zlarfb*.

4.17.745 zlarfg

zlarfg generates a complex elementary reflector H of order n, such that

```
H**H * ( alpha ) = ( beta ),   H**H * H = I.
       (   x   )   (   0  )
```

where alpha and beta are scalars, with beta real, and x is an (n-1)-element complex vector. H is represented in the form

```
H = I - tau * ( 1 ) * ( 1 v**H ) ,
              ( v )
```

where tau is a complex scalar and v is a complex (n-1)-element vector. Note that H is not hermitian.

If the elements of x are all zero and alpha is real, then tau = 0 and H is taken to be the unit matrix.

Otherwise 1 <= real(tau) <= 2 and abs(tau-1) <= 1 .

Syntax

Fortran specification:

```
use armpl_library

subroutine zlarfg(N, ALPHA, X, INCX, TAU)
```

C specification:

```
#include "armpl.h"

void zlarfg_(const armpl_int_t *n, armpl_doublecomplex_t *alpha,
             armpl_doublecomplex_t *x, const armpl_int_t *incx,
             armpl_doublecomplex_t *tau);
```

Parameters

N Input parameter.

N is INTEGER

The order of the elementary reflector.

ALPHA Input and output parameter.

ALPHA is COMPLEX*16

On entry, the value alpha. On exit, it is overwritten with the value beta.

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension. (1+(N-2)*abs(INCX)) On entry, the vector x. On exit, it is overwritten with the vector v.

INCX Input parameter.

INCX is INTEGER

The increment between elements of X. INCX > 0.

TAU Output parameter.

TAU is COMPLEX*16

The value tau.

Related Information

For this routine in other precisions, please see *clarfg*, *dlarfg* and *slarfg*. It also exists with a native C interface as *LAPACKE_zlarfg*.

4.17.746 zlarfgp

zlarfgp generates a complex elementary reflector H of order n, such that

```
H**H * ( alpha ) = ( beta ),   H**H * H = I.
       (   x   )   (   0  )
```

where alpha and beta are scalars, beta is real and non-negative, and x is an (n-1)-element complex vector. H is represented in the form

```
H = I - tau * ( 1 ) * ( 1 v**H ) ,
              ( v )
```

where tau is a complex scalar and v is a complex (n-1)-element vector. Note that H is not hermitian.

If the elements of x are all zero and alpha is real, then tau = 0 and H is taken to be the unit matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlarfgp(N, ALPHA, X, INCX, TAU)
```

C specification:

```
#include "armpl.h"

void zlarfgp_(const armpl_int_t *n, armpl_doublecomplex_t *alpha,
              armpl_doublecomplex_t *x, const armpl_int_t *incx,
              armpl_doublecomplex_t *tau);
```

Parameters

N Input parameter.

N is INTEGER

The order of the elementary reflector.

ALPHA Input and output parameter.

ALPHA is COMPLEX*16

On entry, the value alpha. On exit, it is overwritten with the value beta.

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension. (1+(N-2)*abs(INCX)) On entry, the vector x. On exit, it is overwritten with the vector v.

INCX Input parameter.

INCX is INTEGER

The increment between elements of X. INCX > 0.

TAU Output parameter.

TAU is COMPLEX*16

The value tau.

Related Information

For this routine in other precisions, please see *clarfgp*, *dlarfgp* and *slarfgp*.

4.17.747 zlarft

`zlarft` forms the triangular factor T of a complex block reflector H of order n, which is defined as a product of k elementary reflectors.

If DIRECT = 'F', H = H(1) H(2) . . . H(k) and T is upper triangular;

If DIRECT = 'B', H = H(k) . . . H(2) H(1) and T is lower triangular.

If STOREV = 'C', the vector which defines the elementary reflector H(i) is stored in the i-th column of the array V, and

```
H  =  I - V * T * V**H
```

If STOREV = 'R', the vector which defines the elementary reflector H(i) is stored in the i-th row of the array V, and

```
H  =  I - V**H * T * V
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zlarft(DIRECT, STOREV, N, K, V, LDV, TAU, T, LDT)
```

C specification:

```
#include "armpl.h"

void zlarft_(const char *direct, const char *storev, const armpl_int_t *n,
             const armpl_int_t *k, const armpl_doublecomplex_t *v,
             const armpl_int_t *ldv, const armpl_doublecomplex_t *tau,
             armpl_doublecomplex_t *t, const armpl_int_t *ldt, ... );
```

Parameters

DIRECT Input parameter.

DIRECT is CHARACTER*1

Specifies the order in which the elementary reflectors are multiplied to form the block reflector: = 'F': H = H(1) H(2) . . . H(k) (Forward) = 'B': H = H(k) . . . H(2) H(1) (Backward)

STOREV Input parameter.

STOREV is CHARACTER*1

Specifies how the vectors which define the elementary reflectors are stored (see also Further Details): = 'C': columnwise = 'R': rowwise

N Input parameter.

N is INTEGER

The order of the block reflector H. N >= 0.

K Input parameter.

K is INTEGER

The order of the triangular factor T (= the number of elementary reflectors). K >= 1.

V Input parameter.

V is COMPLEX*16

V is an array, dimension. (LDV, K) if STOREV = 'C' (LDV, N) if STOREV = 'R' The matrix V. See further details.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. If STOREV = 'C', LDV >= max(1, N); if STOREV = 'R', LDV >= K.

TAU Input parameter.

TAU is COMPLEX*16

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i).

T Output parameter.

T is COMPLEX*16

T is an array, dimension (LDT, K). The k by k triangular factor T of the block reflector. If DIRECT = 'F', T is upper triangular; if DIRECT = 'B', T is lower triangular. The rest of the array is not used.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= K.

Related Information

For this routine in other precisions, please see *clarft*, *dlarft* and *slarft*. It also exists with a native C interface as *LAPACKE_zlarft*.

4.17.748 zlarfx

zlarfx applies a complex elementary reflector H to a complex m by n matrix C, from either the left or the right. H is represented in the form

```
H = I - tau * v * v**H
```

where tau is a complex scalar and v is a complex vector.

If tau = 0, then H is taken to be the unit matrix

This version uses inline code if H has order < 11.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlarfx(SIDE, M, N, V, TAU, C, LDC, WORK)
```

C specification:

```
#include "armpl.h"

void zlarfx_(const char *side, const armpl_int_t *m, const armpl_int_t *n,
             const armpl_doublecomplex_t *v, const armpl_doublecomplex_t *tau,
             armpl_doublecomplex_t *c, const armpl_int_t *ldc,
             armpl_doublecomplex_t *work, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': form H * C = 'R': form C * H

M Input parameter.

M is INTEGER

The number of rows of the matrix C.

N Input parameter.

N is INTEGER

The number of columns of the matrix C.

V Input parameter.

V is COMPLEX*16

V is an array, dimension (M) if SIDE = 'L'. or (N) if SIDE = 'R' The vector v in the representation of H.

TAU Input parameter.

TAU is COMPLEX*16

The value tau in the representation of H.

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). On entry, the m by n matrix C. On exit, C is overwritten by the matrix H * C if SIDE = 'L', or C * H if SIDE = 'R'.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDA >= max(1, M).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N) if SIDE = 'L'. or (M) if SIDE = 'R' WORK is not referenced if H has order < 11.

Related Information

For this routine in other precisions, please see *clarfx*, *dlarfx* and *slarfx*. It also exists with a native C interface as *LAPACKE_zlarfx*.

4.17.749 zlarfy

zlarfy applies an elementary reflector, or Householder matrix, H, to an n x n Hermitian matrix C, from both the left and the right.

H is represented in the form

```
H = I - tau * v * v'
```

where tau is a scalar and v is a vector.

If tau is zero, then H is taken to be the unit matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlarfy(UPLO, N, V, INCV, TAU, C, LDC, WORK)
```

C specification:

```
#include "armpl.h"

void zlarfy_(const char *uplo, const armpl_int_t *n,
             const armpl_doublecomplex_t *v, const armpl_int_t *incv,
             const armpl_doublecomplex_t *tau, armpl_doublecomplex_t *c,
             const armpl_int_t *ldc, armpl_doublecomplex_t *work, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the Hermitian matrix C is stored. = 'U': Upper triangle = 'L': Lower triangle

N Input parameter.

N is INTEGER

The number of rows and columns of the matrix C. N >= 0.

V Input parameter.

V is COMPLEX*16

V is an array, dimension. (1 + (N-1)*abs(INCV)) The vector v as described above.

INCV Input parameter.

INCV is INTEGER

The increment between successive elements of v. INCV must not be zero.

TAU Input parameter.

TAU is COMPLEX*16

The value tau as described above.

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). On entry, the matrix C. On exit, C is overwritten by H * C * H'.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, N).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N) .

Related Information

For this routine in other precisions, please see *clarfy*, *dlarfy* and *slarfy*.

4.17.750 zlargv

zlargv generates a vector of complex plane rotations with real cosines, determined by elements of the complex vectors x and y. For i = 1,2,...,n

```
(        c(i)   s(i) ) ( x(i) ) = ( r(i) )
( -conjg(s(i))  c(i) ) ( y(i) ) = (   0  )

where c(i)**2 + ABS(s(i))**2 = 1
```

The following conventions are used (these are the same as in ZLARTG, but differ from the BLAS1 routine ZROTG):

```
If y(i)=0, then c(i)=1 and s(i)=0.
If x(i)=0, then c(i)=0 and s(i) is chosen so that r(i) is real.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zlargv(N, X, INCX, Y, INCY, C, INCC)
```

C specification:

```
#include "armpl.h"

void zlargv_(const armpl_int_t *n, armpl_doublecomplex_t *x,
             const armpl_int_t *incx, armpl_doublecomplex_t *y,
             const armpl_int_t *incy, double *c, const armpl_int_t *incc);
```

Parameters

N Input parameter.

N is INTEGER

The number of plane rotations to be generated.

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension (1+(N-1)*INCX). On entry, the vector x. On exit, x(i) is overwritten by r(i), for i = 1,...,n.

INCX Input parameter.

INCX is INTEGER

The increment between elements of X. INCX > 0.

Y Input and output parameter.

Y is COMPLEX*16

Y is an array, dimension (1+(N-1)*INCY). On entry, the vector y. On exit, the sines of the plane rotations.

INCY Input parameter.

INCY is INTEGER

The increment between elements of Y. INCY > 0.

C Output parameter.

C is DOUBLE PRECISION

C is an array, dimension (1+(N-1)*INCC). The cosines of the plane rotations.

INCC Input parameter.

INCC is INTEGER

The increment between elements of C. INCC > 0.

Related Information

For this routine in other precisions, please see *clargv*, *dlargv* and *slargv*.

4.17.751 zlarnv

zlarnv returns a vector of n random complex numbers from a uniform or normal distribution.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlarnv(IDIST, ISEED, N, X)
```

C specification:

```
#include "armpl.h"

void zlarnv_(const armpl_int_t *idist, armpl_int_t *iseed,
             const armpl_int_t *n, armpl_doublecomplex_t *x);
```

Parameters

IDIST Input parameter.

IDIST is INTEGER

Specifies the distribution of the random numbers: = 1: real and imaginary parts each uniform (0,1) = 2: real and imaginary parts each uniform (-1,1) = 3: real and imaginary parts each normal (0,1) = 4: uniformly distributed on the disc abs(z) < 1 = 5: uniformly distributed on the circle abs(z) = 1

ISEED Input and output parameter.

ISEED is INTEGER array, dimension (4)

On entry, the seed of the random number generator; the array elements must be between 0 and 4095, and ISEED(4) must be odd. On exit, the seed is updated.

N Input parameter.

N is INTEGER

The number of random numbers to be generated.

X Output parameter.

X is COMPLEX*16

X is an array, dimension (N). The generated random numbers.

Related Information

For this routine in other precisions, please see *clarnv*, *dlarnv* and *slarnv*. It also exists with a native C interface as *LAPACKE_zlarnv*.

4.17.752 zlarrv

zlarrv computes the eigenvectors of the tridiagonal matrix T = L D L^T given L, D and APPROXIMATIONS to the eigenvalues of L D L^T . The input eigenvalues should have been computed by DLARRE.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlarrv(N, VL, VU, D, L, PIVMIN, ISPLIT, M, DOL, DOU, MINRGP, RTOL1,
                  RTOL2, W, WERR, WGAP, IBLOCK, INDEXW, GERS, Z, LDZ, ISUPPZ,
                  WORK, IWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zlarrv_(const armpl_int_t *n, const double *vl, const double *vu,
             double *d, double *l, double *pivmin, const armpl_int_t *isplit,
             const armpl_int_t *m, const armpl_int_t *dol,
             const armpl_int_t *dou, const double *minrgp,
             const double *rtol1, const double *rtol2, double *w,
             double *werr, double *wgap, const armpl_int_t *iblock,
             const armpl_int_t *indexw, const double *gers,
             armpl_doublecomplex_t *z, const armpl_int_t *ldz,
             armpl_int_t *isuppz, double *work, armpl_int_t *iwork,
             armpl_int_t *info);
```

Parameters

N Input parameter.

N is INTEGER

The order of the matrix. N >= 0.

VL Input parameter.

VL is DOUBLE PRECISION

Lower bound of the interval that contains the desired eigenvalues. VL < VU. Needed to compute gaps on the left or right end of the extremal eigenvalues in the desired RANGE.

VU Input parameter.

VU is DOUBLE PRECISION

Upper bound of the interval that contains the desired eigenvalues. VL < VU. Needed to compute gaps on the left or right end of the extremal eigenvalues in the desired RANGE.

D Input and output parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). On entry, the N diagonal elements of the diagonal matrix D. On exit, D may be overwritten.

L Input and output parameter.

L is DOUBLE PRECISION

L is an array, dimension (N). On entry, the (N-1) subdiagonal elements of the unit bidiagonal matrix L are in elements 1 to N-1 of L (if the matrix is not split.) At the end of each block is stored the corresponding shift as given by DLARRE. On exit, L is overwritten.

PIVMIN Input parameter.

PIVMIN is DOUBLE PRECISION

The minimum pivot allowed in the Sturm sequence.

ISPLIT Input parameter.

ISPLIT is INTEGER array, dimension (N)

The splitting points, at which T breaks up into blocks. The first block consists of rows/columns 1 to ISPLIT(1), the second of rows/columns ISPLIT(1)+1 through ISPLIT(2), etc.

M Input parameter.

M is INTEGER

The total number of input eigenvalues. 0 <= M <= N.

DOL Input parameter.

DOL is INTEGER

DOU Input parameter.

DOU is INTEGER

If the user wants to compute only selected eigenvectors from all the eigenvalues supplied, he can specify an index range DOL:DOU. Or else the setting DOL=1, DOU=M should be applied. Note that DOL and DOU refer to the order in which the eigenvalues are stored in W. If the user wants to compute only selected eigenpairs, then the columns DOL-1 to DOU+1 of the eigenvector space Z contain the computed eigenvectors. All other columns of Z are set to zero.

MINRGP Input parameter.

MINRGP is DOUBLE PRECISION

RTOL1 Input parameter.

RTOL1 is DOUBLE PRECISION

RTOL2 Input parameter.

RTOL2 is DOUBLE PRECISION

Parameters for bisection. An interval [LEFT,RIGHT] has converged if RIGHT-LEFT.LT.MAX(RTOL1*GAP, RTOL2*MAX(|LEFT|,|RIGHT|))

W Input and output parameter.

W is DOUBLE PRECISION

W is an array, dimension (N). The first M elements of W contain the APPROXIMATE eigenvalues for which eigenvectors are to be computed. The eigenvalues should be grouped by split-off block and ordered from smallest to largest within the block (The output array W from DLARRE is expected here). Furthermore, they are with respect to the shift of the corresponding root representation for their block. On exit, W holds the eigenvalues of the UNshifted matrix.

WERR Input and output parameter.

WERR is DOUBLE PRECISION

WERR is an array, dimension (N). The first M elements contain the semiwidth of the uncertainty interval of the corresponding eigenvalue in W

WGAP Input and output parameter.

WGAP is DOUBLE PRECISION

WGAP is an array, dimension (N). The separation from the right neighbor eigenvalue in W.

IBLOCK Input parameter.

IBLOCK is INTEGER array, dimension (N)

The indices of the blocks (submatrices) associated with the corresponding eigenvalues in W; IBLOCK(i)=1 if eigenvalue W(i) belongs to the first block from the top, =2 if W(i) belongs to the second block, etc.

INDEXW Input parameter.

INDEXW is INTEGER array, dimension (N)

The indices of the eigenvalues within each block (submatrix); for example, INDEXW(i)= 10 and IBLOCK(i)=2 imply that the i-th eigenvalue W(i) is the 10-th eigenvalue in the second block.

GERS Input parameter.

GERS is DOUBLE PRECISION

GERS is an array, dimension (2*N). The N Gerschgorin intervals (the i-th Gerschgorin interval is (GERS(2*i-1), GERS(2*i)). The Gerschgorin intervals should be computed from the original UNshifted matrix.

Z Output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, max(1, M)). If INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix T corresponding to the input eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i). Note: the user must ensure that at least max(1, M) columns are supplied in the array Z.

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1, N).

ISUPPZ Output parameter.

ISUPPZ is INTEGER array, dimension (2*max(1, M))

The support of the eigenvectors in Z, i.e., the indices indicating the nonzero elements in Z. The I-th eigenvector is nonzero only in elements ISUPPZ(2*I-1) through ISUPPZ(2*I).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (12*N) .

IWORK Output parameter.

IWORK is INTEGER array, dimension (7*N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit

> 0: A problem occurred in ZLARRV. < 0: One of the called subroutines signaled an internal problem. Needs inspection of the corresponding parameter IINFO for further information.

=-1: Problem in DLARRB when refining a child's eigenvalues. =-2: Problem in DLARRF when computing the RRR of a child. When a child is inside a tight cluster, it can be difficult to find an RRR. A partial remedy from the user's point of view is to make the parameter MINRGP smaller and recompile. However, as the orthogonality of the computed vectors is proportional to 1/MINRGP, the user should be aware that he might be trading in precision when he decreases MINRGP. =-3: Problem in DLARRB when refining a single eigenvalue after the Rayleigh correction was rejected. = 5: The Rayleigh Quotient Iteration failed to converge to full accuracy in MAXITR steps.

Related Information

For this routine in other precisions, please see *clarrv*, *dlarrv* and *slarrv*.

4.17.753 zlarscl2

ZLARSCL2 performs a reciprocal diagonal scaling on an vector:

```
x <-- inv(D) * x
```

where the DOUBLE PRECISION diagonal matrix D is stored as a vector.

Eventually to be replaced by BLAS_zge_diag_scale in the new BLAS standard.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlarscl2(M, N, D, X, LDX)
```

C specification:

```
#include "armpl.h"

void zlarscl2_(const armpl_int_t *m, const armpl_int_t *n, const double *d,
               armpl_doublecomplex_t *x, const armpl_int_t *ldx);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of D and X. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of X. N >= 0.

D Input parameter.

D is DOUBLE PRECISION

D is an array, length M. Diagonal matrix D, stored as a vector of length M.

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, N). On entry, the vector X to be scaled by D. On exit, the scaled vector.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the vector X. LDX >= M.

Related Information

For this routine in other precisions, please see *clarscl2*, *dlarscl2* and *slarscl2*.

4.17.754 zlartg

zlartg generates a plane rotation so that

```
[  CS  SN  ]     [ F ]     [ R ]
[  __      ]  .  [   ]  =  [   ]   where CS**2 + |SN|**2 = 1.
[ -SN  CS  ]     [ G ]     [ 0 ]
```

This is a faster version of the BLAS1 routine ZROTG, except for the following differences:

```
 F and G are unchanged on return.
If G=0, then CS=1 and SN=0.
If F=0, then CS=0 and SN is chosen so that R is real.
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zlartg(F, G, CS, SN, R)
```

C specification:

```
#include "armpl.h"

void zlartg_(const armpl_doublecomplex_t *f, const armpl_doublecomplex_t *g,
             double *cs, armpl_doublecomplex_t *sn,
             armpl_doublecomplex_t *r);
```

Parameters

F Input parameter.

F is COMPLEX*16

The first component of vector to be rotated.

G Input parameter.

G is COMPLEX*16

The second component of vector to be rotated.

CS Output parameter.

CS is DOUBLE PRECISION

The cosine of the rotation.

SN Output parameter.

SN is COMPLEX*16

The sine of the rotation.

R Output parameter.

R is COMPLEX*16

The nonzero component of the rotated vector.

Related Information

For this routine in other precisions, please see *clartg*, *dlartg* and *slartg*.

4.17.755 zlartv

zlartv applies a vector of complex plane rotations with real cosines to elements of the complex vectors x and y. For i = 1,2,...,n

```
( x(i) ) := (         c(i)   s(i) ) ( x(i) )
( y(i) )    ( -conjg(s(i))   c(i) ) ( y(i) )
```

Syntax

Fortran specification:

```
use armpl_library

subroutine zlartv(N, X, INCX, Y, INCY, C, S, INCC)
```

C specification:

```
#include "armpl.h"

void zlartv_(const armpl_int_t *n, armpl_doublecomplex_t *x,
             const armpl_int_t *incx, armpl_doublecomplex_t *y,
             const armpl_int_t *incy, const double *c,
             const armpl_doublecomplex_t *s, const armpl_int_t *incc);
```

Parameters

N Input parameter.

N is INTEGER

The number of plane rotations to be applied.

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension (1+(N-1)*INCX). The vector x.

INCX Input parameter.

INCX is INTEGER

The increment between elements of X. INCX > 0.

Y Input and output parameter.

Y is COMPLEX*16

Y is an array, dimension (1+(N-1)*INCY). The vector y.

INCY Input parameter.

INCY is INTEGER

The increment between elements of Y. INCY > 0.

C Input parameter.

C is DOUBLE PRECISION

C is an array, dimension (1+(N-1)*INCC). The cosines of the plane rotations.

S Input parameter.

S is COMPLEX*16

S is an array, dimension (1+(N-1)*INCC). The sines of the plane rotations.

INCC Input parameter.

INCC is INTEGER

The increment between elements of C and S. INCC > 0.

Related Information

For this routine in other precisions, please see *clartv*, *dlartv* and *slartv*.

4.17.756 zlarz

zlarz applies a complex elementary reflector H to a complex M-by-N matrix C, from either the left or the right. H is represented in the form

```
H = I - tau * v * v**H
```

where tau is a complex scalar and v is a complex vector.

If tau = 0, then H is taken to be the unit matrix.

To apply H^H (the conjugate transpose of H), supply conjg(tau) instead tau.

H is a product of k elementary reflectors as returned by ZTZRZF.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlarz(SIDE, M, N, L, V, INCV, TAU, C, LDC, WORK)
```

C specification:

```
#include "armpl.h"

void zlarz_(const char *side, const armpl_int_t *m, const armpl_int_t *n,
            const armpl_int_t *l, const armpl_doublecomplex_t *v,
            const armpl_int_t *incv, const armpl_doublecomplex_t *tau,
            armpl_doublecomplex_t *c, const armpl_int_t *ldc,
            armpl_doublecomplex_t *work, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': form H * C = 'R': form C * H

M Input parameter.

M is INTEGER

The number of rows of the matrix C.

N Input parameter.

N is INTEGER

The number of columns of the matrix C.

L Input parameter.

L is INTEGER

The number of entries of the vector V containing the meaningful part of the Householder vectors. If SIDE = 'L', M >= L >= 0, if SIDE = 'R', N >= L >= 0.

V Input parameter.

V is COMPLEX*16

V is an array, dimension (1+(L-1)*abs(INCV)). The vector v in the representation of H as returned by ZTZRZF. V is not used if TAU = 0.

INCV Input parameter.

INCV is INTEGER

The increment between elements of v. INCV <> 0.

TAU Input parameter.

TAU is COMPLEX*16

The value tau in the representation of H.

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by the matrix H * C if SIDE = 'L', or C * H if SIDE = 'R'.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension. (N) if SIDE = 'L' or (M) if SIDE = 'R'

Related Information

For this routine in other precisions, please see *clarz*, *dlarz* and *slarz*.

4.17.757 zlarzb

zlarzb applies a complex block reflector H or its transpose H^H to a complex distributed M-by-N C from the left or the right.

Currently, only STOREV = 'R' and DIRECT = 'B' are supported.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlarzb(SIDE, TRANS, DIRECT, STOREV, M, N, K, L, V, LDV, T, LDT, C,
                  LDC, WORK, LDWORK)
```

C specification:

```
#include "armpl.h"

void zlarzb_(const char *side, const char *trans, const char *direct,
             const char *storev, const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *k, const armpl_int_t *l,
             const armpl_doublecomplex_t *v, const armpl_int_t *ldv,
             const armpl_doublecomplex_t *t, const armpl_int_t *ldt,
             armpl_doublecomplex_t *c, const armpl_int_t *ldc,
             armpl_doublecomplex_t *work, const armpl_int_t *ldwork, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply H or H^H from the Left = 'R': apply H or H^H from the Right

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply H (No transpose) = 'C': apply H^H (Conjugate transpose)

DIRECT Input parameter.

DIRECT is CHARACTER*1

Indicates how H is formed from a product of elementary reflectors = 'F': H = H(1) H(2) . . . H(k) (Forward, not supported yet) = 'B': H = H(k) . . . H(2) H(1) (Backward)

STOREV Input parameter.

STOREV is CHARACTER*1

Indicates how the vectors which define the elementary reflectors are stored: = 'C': Columnwise (not supported yet) = 'R': Rowwise

M Input parameter.

M is INTEGER

The number of rows of the matrix C.

N Input parameter.

N is INTEGER

The number of columns of the matrix C.

K Input parameter.

K is INTEGER

The order of the matrix T (= the number of elementary reflectors whose product defines the block reflector).

L Input parameter.

L is INTEGER

The number of columns of the matrix V containing the meaningful part of the Householder reflectors. If SIDE = 'L', M >= L >= 0, if SIDE = 'R', N >= L >= 0.

V Input parameter.

V is COMPLEX*16

V is an array, dimension (LDV,NV).. If STOREV = 'C', NV = K; if STOREV = 'R', NV = L.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. If STOREV = 'C', LDV >= L; if STOREV = 'R', LDV >= K.

T Input parameter.

T is COMPLEX*16

T is an array, dimension (LDT, K). The triangular K-by-K matrix T in the representation of the block reflector.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= K.

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). On entry, the M-by-N matrix C. On exit, C is overwritten by H* C or H^H * C or C*H or C*H^H .

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (LDWORK, K) .

LDWORK Input parameter.

LDWORK is INTEGER

The leading dimension of the array WORK. If SIDE = 'L', LDWORK >= max(1, N); if SIDE = 'R', LDWORK >= max(1, M).

Related Information

For this routine in other precisions, please see *clarzb*, *dlarzb* and *slarzb*.

4.17.758 zlarzt

zlarzt forms the triangular factor T of a complex block reflector H of order > n, which is defined as a product of k elementary reflectors.

If DIRECT = 'F', H = H(1) H(2) . . . H(k) and T is upper triangular;

If DIRECT = 'B', H = H(k) . . . H(2) H(1) and T is lower triangular.

If STOREV = 'C', the vector which defines the elementary reflector H(i) is stored in the i-th column of the array V, and

```
H  =  I - V * T * V**H
```

If STOREV = 'R', the vector which defines the elementary reflector H(i) is stored in the i-th row of the array V, and

```
H  =  I - V**H * T * V
```

Currently, only STOREV = 'R' and DIRECT = 'B' are supported.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlarzt(DIRECT, STOREV, N, K, V, LDV, TAU, T, LDT)
```

C specification:

```
#include "armpl.h"

void zlarzt_(const char *direct, const char *storev, const armpl_int_t *n,
             const armpl_int_t *k, armpl_doublecomplex_t *v,
             const armpl_int_t *ldv, const armpl_doublecomplex_t *tau,
             armpl_doublecomplex_t *t, const armpl_int_t *ldt, ... );
```

Parameters

DIRECT Input parameter.

DIRECT is CHARACTER*1

Specifies the order in which the elementary reflectors are multiplied to form the block reflector: = 'F': H = H(1) H(2) . . . H(k) (Forward, not supported yet) = 'B': H = H(k) . . . H(2) H(1) (Backward)

STOREV Input parameter.

STOREV is CHARACTER*1

Specifies how the vectors which define the elementary reflectors are stored (see also Further Details): = 'C': columnwise (not supported yet) = 'R': rowwise

N Input parameter.

N is INTEGER

The order of the block reflector H. N >= 0.

K Input parameter.

K is INTEGER

The order of the triangular factor T (= the number of elementary reflectors). K >= 1.

V Input and output parameter.

V is COMPLEX*16

V is an array, dimension. (LDV, K) if STOREV = 'C' (LDV, N) if STOREV = 'R' The matrix V. See further details.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. If STOREV = 'C', LDV >= max(1, N); if STOREV = 'R', LDV >= K.

TAU Input parameter.

TAU is COMPLEX*16

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i).

T Output parameter.

T is COMPLEX*16

T is an array, dimension (LDT, K). The k by k triangular factor T of the block reflector. If DIRECT = 'F', T is upper triangular; if DIRECT = 'B', T is lower triangular. The rest of the array is not used.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= K.

Related Information

For this routine in other precisions, please see *clarzt*, *dlarzt* and *slarzt*.

4.17.759 zlascl

zlascl multiplies the M by N complex matrix A by the real scalar CTO/CFROM. This is done without over/underflow as long as the final result CTO*A(I,J)/CFROM does not over/underflow. TYPE specifies that A may be full, upper triangular, lower triangular, upper Hessenberg, or banded.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlascl(TYPE, KL, KU, CFROM, CTO, M, N, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void zlascl_(const char *type, const armpl_int_t *kl, const armpl_int_t *ku,
             const double *cfrom, const double *cto, const armpl_int_t *m,
             const armpl_int_t *n, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *info, ... );
```

Parameters

TYPE Input parameter.

TYPE is CHARACTER*1

TYPE indices the storage type of the input matrix. = 'G': A is a full matrix. = 'L': A is a lower triangular matrix. = 'U': A is an upper triangular matrix. = 'H': A is an upper Hessenberg matrix. = 'B': A is a symmetric band matrix with lower bandwidth KL and upper bandwidth KU and with the only the lower half stored. = 'Q': A is a symmetric band matrix with lower bandwidth KL and upper bandwidth KU and with the only the upper half stored. = 'Z': A is a band matrix with lower bandwidth KL and upper bandwidth KU. See ZGBTRF for storage details.

KL Input parameter.

KL is INTEGER

The lower bandwidth of A. Referenced only if TYPE = 'B', 'Q' or 'Z'.

KU Input parameter.

KU is INTEGER

The upper bandwidth of A. Referenced only if TYPE = 'B', 'Q' or 'Z'.

CFROM Input parameter.

CFROM is DOUBLE PRECISION

CTO Input parameter.

CTO is DOUBLE PRECISION

The matrix A is multiplied by CTO/CFROM. A(I,J) is computed without over/underflow if the final result CTO*A(I,J)/CFROM can be represented without over/underflow. CFROM must be nonzero.

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The matrix to be multiplied by CTO/CFROM. See TYPE for the storage type.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If TYPE = 'G', 'L', 'U', 'H', LDA >= max(1, M); TYPE = 'B', LDA >= KL+1; TYPE = 'Q', LDA >= KU+1; TYPE = 'Z', LDA >= 2*KL+KU+1.

INFO Output parameter.

INFO is INTEGER

0 - successful exit <0 - if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *clascl*, *dlascl* and *slascl*. It also exists with a native C interface as *LAPACKE_zlascl*.

4.17.760 zlascl2

zlascl2 performs a diagonal scaling on a vector:

```
x <-- D * x
```

where the DOUBLE PRECISION diagonal matrix D is stored as a vector.

Eventually to be replaced by BLAS_zge_diag_scale in the new BLAS standard.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlascl2(M, N, D, X, LDX)
```

C specification:

```
#include "armpl.h"

void zlascl2_(const armpl_int_t *m, const armpl_int_t *n, const double *d,
              armpl_doublecomplex_t *x, const armpl_int_t *ldx);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of D and X. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of X. N >= 0.

D Input parameter.

D is DOUBLE PRECISION

D is an array, length M. Diagonal matrix D, stored as a vector of length M.

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension (LDX, N). On entry, the vector X to be scaled by D. On exit, the scaled vector.

LDX Input parameter.

LDX is INTEGER

The leading dimension of the vector X. LDX >= M.

Related Information

For this routine in other precisions, please see *clascl2*, *dlascl2* and *slascl2*.

4.17.761 zlaset

zlaset initializes a 2-D array A to BETA on the diagonal and ALPHA on the offdiagonals.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlaset(UPLO, M, N, ALPHA, BETA, A, LDA)
```

C specification:

```
#include "armpl.h"

void zlaset_(const char *uplo, const armpl_int_t *m, const armpl_int_t *n,
             const armpl_doublecomplex_t *alpha,
             const armpl_doublecomplex_t *beta, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies the part of the matrix A to be set. = 'U': Upper triangular part is set. The lower triangle is unchanged. = 'L': Lower triangular part is set. The upper triangle is unchanged. Otherwise: All of the matrix A is set.

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of A.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of A.

ALPHA Input parameter.

ALPHA is COMPLEX*16

All the offdiagonal array elements are set to ALPHA.

BETA Input parameter.

BETA is COMPLEX*16

All the diagonal array elements are set to BETA.

A Output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the m by n matrix A. On exit, A(i,j) = ALPHA, 1 <= i <= m, 1 <= j <= n, i.ne.j; A(i,i) = BETA , 1 <= i <= min(m,n)

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

Related Information

For this routine in other precisions, please see *claset*, *dlaset* and *slaset*. It also exists with a native C interface as *LAPACKE_zlaset*.

4.17.762 zlasr

zlasr applies a sequence of real plane rotations to a complex matrix A, from either the left or the right.

When SIDE = 'L', the transformation takes the form

```
A := P*A
```

and when SIDE = 'R', the transformation takes the form

```
A := A*P**T
```

where P is an orthogonal matrix consisting of a sequence of z plane rotations, with z = M when SIDE = 'L' and z = N when SIDE = 'R', and P^T is the transpose of P.

When DIRECT = 'F' (Forward sequence), then

```
P = P(z-1) * ... * P(2) * P(1)
```

and when DIRECT = 'B' (Backward sequence), then

```
P = P(1) * P(2) * ... * P(z-1)
```

where P(k) is a plane rotation matrix defined by the 2-by-2 rotation

```
R(k) = (  c(k)   s(k) )
     = ( -s(k)   c(k) ).
```

When PIVOT = 'V' (Variable pivot), the rotation is performed for the plane (k,k+1), i.e., P(k) has the form

```
P(k) = (  1                                          )
       (       ...                                   )
       (              1                              )
       (                    c(k)   s(k)              )
```

(continues on next page)

(continued from previous page)

```
(                   -s(k)   c(k)                    )
(                                1                  )
(                                     ...           )
(                                           1       )
```

where R(k) appears as a rank-2 modification to the identity matrix in rows and columns k and k+1.

When PIVOT = 'T' (Top pivot), the rotation is performed for the plane (1,k+1), so P(k) has the form

```
P(k) = (  c(k)                      s(k)                  )
       (          1                                       )
       (                ...                               )
       (                        1                         )
       ( -s(k)                  c(k)                      )
       (                                1                 )
       (                                     ...          )
       (                                           1      )
```

where R(k) appears in rows and columns 1 and k+1.

Similarly, when PIVOT = 'B' (Bottom pivot), the rotation is performed for the plane (k,z), giving P(k) the form

```
P(k) = ( 1                                              )
       (       ...                                      )
       (               1                                )
       (                     c(k)                s(k)   )
       (                             1                  )
       (                                  ...           )
       (                                        1       )
       (                    -s(k)                c(k)   )
```

where R(k) appears in rows and columns k and z. The rotations are performed without ever forming P(k) explicitly.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlasr(SIDE, PIVOT, DIRECT, M, N, C, S, A, LDA)
```

C specification:

```
#include "armpl.h"

void zlasr_(const char *side, const char *pivot, const char *direct,
            const armpl_int_t *m, const armpl_int_t *n, const double *c,
            const double *s, armpl_doublecomplex_t *a, const armpl_int_t *lda,
            ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

Specifies whether the plane rotation matrix P is applied to A on the left or the right. = 'L': Left, compute A := P*A = 'R': Right, compute A:= A*P^T

PIVOT Input parameter.

PIVOT is CHARACTER*1

Specifies the plane for which P(k) is a plane rotation matrix. = 'V': Variable pivot, the plane (k,k+1) = 'T': Top pivot, the plane (1,k+1) = 'B': Bottom pivot, the plane (k,z)

DIRECT Input parameter.

DIRECT is CHARACTER*1

Specifies whether P is a forward or backward sequence of plane rotations. = 'F': Forward, P = P(z-1)*...*P(2)*P(1) = 'B': Backward, P = P(1)*P(2)*...*P(z-1)

M Input parameter.

M is INTEGER

The number of rows of the matrix A. If m <= 1, an immediate return is effected.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. If n <= 1, an immediate return is effected.

C Input parameter.

C is DOUBLE PRECISION

C is an array, dimension. (M-1) if SIDE = 'L' (N-1) if SIDE = 'R' The cosines c(k) of the plane rotations.

S Input parameter.

S is DOUBLE PRECISION

S is an array, dimension. (M-1) if SIDE = 'L' (N-1) if SIDE = 'R' The sines s(k) of the plane rotations. The 2-by-2 plane rotation part of the matrix P(k), R(k), has the form R(k) = (c(k) s(k)) (-s(k) c(k)).

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The M-by-N matrix A. On exit, A is overwritten by P* A if SIDE = 'R' or by A*P^T if SIDE = 'L'.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

Related Information

For this routine in other precisions, please see *clasr*, *dlasr* and *slasr*.

4.17.763 zlassq

zlassq returns the values scl and ssq such that

```
( scl**2 )*ssq = x( 1 )**2 +...+ x( n )**2 + ( scale**2 )*sumsq,
```

where x(i) = abs(X(1 + (i - 1)*INCX)). The value of sumsq is assumed to be at least unity and the value of ssq will then satisfy

```
1.0 .le. ssq .le. ( sumsq + 2*n ).
```

scale is assumed to be non-negative and scl returns the value

```
scl = max( scale, abs( real( x( i ) ) ), abs( aimag( x( i ) ) ) ),
      i
```

scale and sumsq must be supplied in SCALE and SUMSQ respectively. SCALE and SUMSQ are overwritten by scl and ssq respectively.

The routine makes only one pass through the vector X.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlassq(N, X, INCX, SCALE, SUMSQ)
```

C specification:

```
#include "armpl.h"

void zlassq_(const armpl_int_t *n, const armpl_doublecomplex_t *x,
             const armpl_int_t *incx, double *scale, double *sumsq);
```

Parameters

N Input parameter.

N is INTEGER

The number of elements to be used from the vector X.

X Input parameter.

X is COMPLEX*16

X is an array, dimension (N). The vector x as described above. x(i) = X(1 + (i - 1)* INCX), 1 <= i <= n.

INCX Input parameter.

INCX is INTEGER

The increment between successive values of the vector X. INCX > 0.

SCALE Input and output parameter.

SCALE is DOUBLE PRECISION

On entry, the value scale in the equation above. On exit, SCALE is overwritten with the value scl .

SUMSQ Input and output parameter.

SUMSQ is DOUBLE PRECISION

On entry, the value sumsq in the equation above. On exit, SUMSQ is overwritten with the value ssq .

Related Information

For this routine in other precisions, please see *classq*, *dlassq* and *slassq*. It also exists with a native C interface as *LAPACKE_zlassq*.

4.17.764 zlaswlq

zlaswlq computes a blocked Short-Wide LQ factorization of a M-by-N matrix A, where N >= M: A = L * Q

Syntax

Fortran specification:

```
use armpl_library

subroutine zlaswlq(M, N, MB, NB, A, LDA, T, LDT, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zlaswlq_(const armpl_int_t *m, const armpl_int_t *n,
              const armpl_int_t *mb, const armpl_int_t *nb,
              armpl_doublecomplex_t *a, const armpl_int_t *lda,
              armpl_doublecomplex_t *t, const armpl_int_t *ldt,
              armpl_doublecomplex_t *work, const armpl_int_t *lwork,
              armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= M >= 0.

MB Input parameter.

MB is INTEGER

The row block size to be used in the blocked QR. M >= MB >= 1

NB Input parameter.

NB is INTEGER

The column block size to be used in the blocked QR. NB > M.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and below the diagonal of the array contain the N-by-N lower triangular matrix L; the elements above the diagonal represent Q by the rows of blocked V (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

T Output parameter.

T is COMPLEX*16

T is an array,. dimension (LDT, N * Number_of_row_blocks) where Number_of_row_blocks = CEIL((N-M)/(NB-M)) The blocked upper triangular block reflectors stored in compact form as a sequence of upper triangular blocks. See Further Details below.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= MB.

WORK Output parameter.

(workspace) COMPLEX*16

(workspace) is an array, dimension (MAX(1, LWORK)) .

LWORK Input parameter.

The dimension of the array WORK. LWORK >= MB*M.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *claswlq*, *dlaswlq* and *slaswlq*.

4.17.765 zlaswp

zlaswp performs a series of row interchanges on the matrix A. One row interchange is initiated for each of rows K1 through K2 of A.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlaswp(N, A, LDA, K1, K2, IPIV, INCX)
```

C specification:

```
#include "armpl.h"

void zlaswp_(const armpl_int_t *n, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, const armpl_int_t *k1,
             const armpl_int_t *k2, const armpl_int_t *ipiv,
             const armpl_int_t *incx);
```

Parameters

N Input parameter.

N is INTEGER

The number of columns of the matrix A.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the matrix of column dimension N to which the row interchanges will be applied. On exit, the permuted matrix.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A.

K1 Input parameter.

K1 is INTEGER

The first element of IPIV for which a row interchange will be done.

K2 Input parameter.

K2 is INTEGER

(K2-K1+1) is the number of elements of IPIV for which a row interchange will be done.

IPIV Input parameter.

IPIV is INTEGER array, dimension (K1+(K2-K1)*abs(INCX))

The vector of pivot indices. Only the elements in positions K1 through K1+(K2-K1)*abs(INCX) of IPIV are accessed. IPIV(K1+(K-K1)*abs(INCX)) = L implies rows K and L are to be interchanged.

INCX Input parameter.

INCX is INTEGER

The increment between successive values of IPIV. If INCX is negative, the pivots are applied in reverse order.

Related Information

For this routine in other precisions, please see *claswp*, *dlaswp* and *slaswp*. It also exists with a native C interface as *LAPACKE_zlaswp*.

4.17.766 zlasyf

zlasyf computes a partial factorization of a complex symmetric matrix A using the Bunch-Kaufman diagonal pivoting method. The partial factorization has the form:

A = (I U12) (A11 0) (I 0) if UPLO = 'U', or:

```
( 0  U22 ) (  0   D  ) ( U12**T U22**T )
```

A = (L11 0) (D 0) ($L11^T$ $L21^T$) if UPLO = 'L'

```
( L21  I ) ( 0   A22 ) (  0       I    )
```

where the order of D is at most NB. The actual order is returned in the argument KB, and is either NB or NB-1, or N if N <= NB. Note that U^T denotes the transpose of U.

zlasyf is an auxiliary routine called by ZSYTRF. It uses blocked code (calling Level 3 BLAS) to update the submatrix A11 (if UPLO = 'U') or A22 (if UPLO = 'L').

Syntax

Fortran specification:

```
use armpl_library

subroutine zlasyf(UPLO, N, NB, KB, A, LDA, IPIV, W, LDW, INFO)
```

C specification:

```
#include "armpl.h"

void zlasyf_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nb,
             armpl_int_t *kb, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *ipiv,
             armpl_doublecomplex_t *w, const armpl_int_t *ldw,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NB Input parameter.

NB is INTEGER

The maximum number of columns of the matrix A that should be factored. NB should be at least 2 to allow for 2-by-2 pivot blocks.

KB Output parameter.

KB is INTEGER

The number of columns of A that were actually factored. KB is either NB-1 or NB, or N if N <= NB.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, A contains details of the partial factorization.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D.

If UPLO = 'U': Only the last KB elements of IPIV are set.

If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block.

If UPLO = 'L': Only the first KB elements of IPIV are set.

If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

W Output parameter.

W is COMPLEX*16

W is an array, dimension (LDW, NB) .

LDW Input parameter.

LDW is INTEGER

The leading dimension of the array W. LDW >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit > 0: if INFO = k, D(k,k) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular.

Related Information

For this routine in other precisions, please see *clasyf*, *dlasyf* and *slasyf*.

4.17.767 zlasyf_aa

DLATRF_AA factorizes a panel of a complex symmetric matrix A using the Aasen's algorithm. The panel consists of a set of NB rows of A when UPLO is U, or a set of NB columns when UPLO is L.

In order to factorize the panel, the Aasen's algorithm requires the last row, or column, of the previous panel. The first row, or column, of A is set to be the first row, or column, of an identity matrix, which is used to factorize the first panel.

The resulting J-th row of U, or J-th column of L, is stored in the (J-1)-th row, or column, of A (without the unit diagonals), while the diagonal and subdiagonal of A are overwritten by those of T.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlasyf_aa(UPLO, J1, M, NB, A, LDA, IPIV, H, LDH, WORK)
```

C specification:

```
#include "armpl.h"

void zlasyf_aa_(const char *uplo, const armpl_int_t *j1, const armpl_int_t *m,
                const armpl_int_t *nb, armpl_doublecomplex_t *a,
                const armpl_int_t *lda, armpl_int_t *ipiv,
```

(continues on next page)

(continued from previous page)

```
                armpl_doublecomplex_t *h, const armpl_int_t *ldh,
                armpl_doublecomplex_t *work, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': Upper triangle of A is stored; = 'L': Lower triangle of A is stored.

J1 Input parameter.

J1 is INTEGER

The location of the first row, or column, of the panel within the submatrix of A, passed to this routine, e.g., when called by ZSYTRF_AA, for the first panel, J1 is 1, while for the remaining panels, J1 is 2.

M Input parameter.

M is INTEGER

The dimension of the submatrix. M >= 0.

NB Input parameter.

NB is INTEGER

The dimension of the panel to be facotorized.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, M) for. the first panel, while dimension (LDA,M+1) for the remaining panels.

On entry, A contains the last row, or column, of the previous panel, and the trailing submatrix of A to be factorized, except for the first panel, only the panel is passed.

On exit, the leading panel is factorized.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

IPIV Output parameter.

IPIV is INTEGER array, dimension (M)

Details of the row and column interchanges, the row and column k were interchanged with the row and column IPIV(k).

H Input and output parameter.

H is COMPLEX*16 workspace, dimension (LDH, NB).

LDH Input parameter.

LDH is INTEGER

The leading dimension of the workspace H. LDH >= max(1, M).

WORK Output parameter.

WORK is COMPLEX*16 workspace, dimension (M).

Related Information

For this routine in other precisions, please see *clasyf_aa*, *dlasyf_aa* and *slasyf_aa*.

4.17.768 zlasyf_rook

ZLASYF_ROOK computes a partial factorization of a complex symmetric matrix A using the bounded Bunch-Kaufman ("rook") diagonal pivoting method. The partial factorization has the form:

A = (I U12) (A11 0) (I 0) if UPLO = 'U', or:

```
( 0  U22 ) (  0   D  ) ( U12**T U22**T )
```

A = (L11 0) (D 0) ($L11^T$ $L21^T$) if UPLO = 'L'

```
( L21  I ) (  0  A22 ) (  0       I    )
```

where the order of D is at most NB. The actual order is returned in the argument KB, and is either NB or NB-1, or N if N <= NB.

ZLASYF_ROOK is an auxiliary routine called by ZSYTRF_ROOK. It uses blocked code (calling Level 3 BLAS) to update the submatrix A11 (if UPLO = 'U') or A22 (if UPLO = 'L').

Syntax

Fortran specification:

```
use armpl_library

subroutine zlasyf_rook(UPLO, N, NB, KB, A, LDA, IPIV, W, LDW, INFO)
```

C specification:

```
#include "armpl.h"

void zlasyf_rook_(const char *uplo, const armpl_int_t *n,
                  const armpl_int_t *nb, armpl_int_t *kb,
                  armpl_doublecomplex_t *a, const armpl_int_t *lda,
                  armpl_int_t *ipiv, armpl_doublecomplex_t *w,
                  const armpl_int_t *ldw, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

NB Input parameter.

NB is INTEGER

The maximum number of columns of the matrix A that should be factored. NB should be at least 2 to allow for 2-by-2 pivot blocks.

KB Output parameter.

KB is INTEGER

The number of columns of A that were actually factored. KB is either NB-1 or NB, or N if N <= NB.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit, A contains details of the partial factorization.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D.

If UPLO = 'U': Only the last KB elements of IPIV are set.

If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k-1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k-1 and -IPIV(k-1) were inerchaged, D(k-1:k,k-1:k) is a 2-by-2 diagonal block.

If UPLO = 'L': Only the first KB elements of IPIV are set.

If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k+1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k+1 and -IPIV(k+1) were inerchaged, D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

W Output parameter.

W is COMPLEX*16

W is an array, dimension (LDW, NB) .

LDW Input parameter.

LDW is INTEGER

The leading dimension of the array W. LDW >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit > 0: if INFO = k, D(k,k) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular.

Related Information

For this routine in other precisions, please see *clasyf_rook*, *dlasyf_rook* and *slasyf_rook*.

4.17.769 zlat2c

`zlat2c` converts a COMPLEX*16 triangular matrix, SA, to a COMPLEX triangular matrix, A.

RMAX is the overflow for the SINGLE PRECISION arithmetic `zlat2c` checks that all the entries of A are between -RMAX and RMAX. If not the conversion is aborted and a flag is raised.

This is an auxiliary routine so there is no argument checking.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlat2c(UPLO, N, A, LDA, SA, LDSA, INFO)
```

C specification:

```
#include "armpl.h"

void zlat2c_(const char *uplo, const armpl_int_t *n,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_singlecomplex_t *sa, const armpl_int_t *ldsa,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

= 'U': `A` is upper triangular; = 'L': `A` is lower triangular.

N Input parameter.

`N` is `INTEGER`

The number of rows and columns of the matrix `A`. `N` >= 0.

A Input parameter.

`A` is `COMPLEX*16`

`A` is an array, dimension (`LDA`, `N`). On entry, the N-by-N triangular coefficient matrix `A`.

LDA Input parameter.

`LDA` is `INTEGER`

The leading dimension of the array `A`. `LDA` >= max(1, `N`).

SA Output parameter.

`SA` is `COMPLEX`

`SA` is an array, dimension (`LDSA`, `N`). Only the `UPLO` part of `SA` is referenced. On exit, if INFO=0, the N-by-N coefficient matrix SA; if INFO>0, the content of the `UPLO` part of `SA` is unspecified.

LDSA Input parameter.

`LDSA` is `INTEGER`

The leading dimension of the array `SA`. `LDSA` >= max(1,M).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. = 1: an entry of the matrix A is greater than the SINGLE PRECISION overflow threshold, in this case, the content of the UPLO part of SA in exit is unspecified.

Related Information

4.17.770 zlatbs

zlatbs solves one of the triangular systems

```
A * x = s*b,  A**T * x = s*b,  or  A**H * x = s*b,
```

with scaling to prevent overflow, where A is an upper or lower triangular band matrix. Here A^T denotes the transpose of A, x and b are n-element vectors, and s is a scaling factor, usually less than or equal to 1, chosen so that the components of x will be less than the overflow threshold. If the unscaled problem will not cause overflow, the Level 2 BLAS routine ZTBSV is called. If the matrix A is singular (A(j,j) = 0 for some j), then s is set to 0 and a non-trivial solution to A*x = 0 is returned.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlatbs(UPLO, TRANS, DIAG, NORMIN, N, KD, AB, LDAB, X, SCALE, CNORM,
                  INFO)
```

C specification:

```
#include "armpl.h"

void zlatbs_(const char *uplo, const char *trans, const char *diag,
             const char *normin, const armpl_int_t *n, const armpl_int_t *kd,
             const armpl_doublecomplex_t *ab, const armpl_int_t *ldab,
             armpl_doublecomplex_t *x, double *scale, double *cnorm,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the matrix A is upper or lower triangular. = 'U': Upper triangular = 'L': Lower triangular

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the operation applied to A. = 'N': Solve A * x = s*b (No transpose) = 'T': Solve A^T * x = s*b (Transpose) = 'C': Solve A^H * x = s*b (Conjugate transpose)

DIAG Input parameter.

DIAG is CHARACTER*1

Specifies whether or not the matrix A is unit triangular. = 'N': Non-unit triangular = 'U': Unit triangular

NORMIN Input parameter.

NORMIN is CHARACTER*1

Specifies whether CNORM has been set or not. = 'Y': CNORM contains the column norms on entry = 'N': CNORM is not set on entry. On exit, the norms will be computed and stored in CNORM.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

KD Input parameter.

KD is INTEGER

The number of subdiagonals or superdiagonals in the triangular matrix A. KD >= 0.

AB Input parameter.

AB is COMPLEX*16

AB is an array, dimension (LDAB, N). The upper or lower triangular band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension (N). On entry, the right hand side b of the triangular system. On exit, X is overwritten by the solution vector x.

SCALE Output parameter.

SCALE is DOUBLE PRECISION

The scaling factor s for the triangular system A * x = s*b, A^T * x = s*b, or A^H * x = s*b. If SCALE = 0, the matrix A is singular or badly scaled, and the vector x is an exact or approximate solution to A*x = 0.

CNORM Input and output parameter.

CNORM is DOUBLE PRECISION

CNORM is an array, dimension (N). If NORMIN = 'Y', CNORM is an input argument and CNORM(j) contains the norm of the off-diagonal part of the j-th column of A. If TRANS = 'N', CNORM(j) must be greater than or equal to the infinity-norm, and if TRANS = 'T' or 'C', CNORM(j) must be greater than or equal to the 1-norm.

If NORMIN = 'N', CNORM is an output argument and CNORM(j) returns the 1-norm of the offdiagonal part of the j-th column of A.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value

Related Information

For this routine in other precisions, please see *clatbs*, *dlatbs* and *slatbs*.

4.17.771 zlatdf

`zlatdf` computes the contribution to the reciprocal Dif-estimate by solving for x in Z * x = b, where b is chosen such that the norm of x is as large as possible. It is assumed that LU decomposition of Z has been computed by ZGETC2. On entry RHS = f holds the contribution from earlier solved sub-systems, and on return RHS = x.

The factorization of Z returned by ZGETC2 has the form Z = P * L * U * Q, where P and Q are permutation matrices. L is lower triangular with unit diagonal elements and U is upper triangular.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlatdf(IJOB, N, Z, LDZ, RHS, RDSUM, RDSCAL, IPIV, JPIV)
```

C specification:

```
#include "armpl.h"

void zlatdf_(const armpl_int_t *ijob, const armpl_int_t *n,
             const armpl_doublecomplex_t *z, const armpl_int_t *ldz,
             armpl_doublecomplex_t *rhs, double *rdsum, double *rdscal,
             const armpl_int_t *ipiv, const armpl_int_t *jpiv);
```

Parameters

IJOB Input parameter.

`IJOB` is `INTEGER`

`IJOB` = 2: First compute an approximative null-vector e of `Z` using ZGECON, e is normalized and solve for Zx = +-e - f with the sign giving the greater value of 2-norm(x). About 5 times as expensive as Default. `IJOB` .ne. 2: Local look ahead strategy where all entries of the r.h.s. b is chosen as either +1 or -1. Default.

N Input parameter.

`N` is `INTEGER`

The number of columns of the matrix `Z`.

Z Input parameter.

`Z` is `COMPLEX*16`

`Z` is an array, dimension (`LDZ`, N). On entry, the LU part of the factorization of the n-by-n matrix `Z` computed by ZGETC2: `Z` = P * L * U * Q

LDZ Input parameter.

`LDZ` is `INTEGER`

The leading dimension of the array `Z`. LDA >= max(1, N).

RHS Input and output parameter.

`RHS` is `COMPLEX*16`

`RHS` is an array, dimension (`N`).. On entry, `RHS` contains contributions from other subsystems. On exit, `RHS` contains the solution of the subsystem with entries according to the value of `IJOB` (see above).

RDSUM Input and output parameter.

`RDSUM` is `DOUBLE PRECISION`

On entry, the sum of squares of computed contributions to the Dif-estimate under computation by ZTGSYL, where the scaling factor RDSCAL (see below) has been factored out. On exit, the corresponding sum of squares updated with the contributions from the current sub-system. If TRANS = 'T' RDSUM is not touched. NOTE: RDSUM only makes sense when ZTGSY2 is called by CTGSYL.

RDSCAL Input and output parameter.

RDSCAL is DOUBLE PRECISION

On entry, scaling factor used to prevent overflow in RDSUM. On exit, RDSCAL is updated w.r.t. the current contributions in RDSUM. If TRANS = 'T', RDSCAL is not touched. NOTE: RDSCAL only makes sense when ZTGSY2 is called by ZTGSYL.

IPIV Input parameter.

IPIV is INTEGER array, dimension (N).

The pivot indices; for 1 <= i <= N, row i of the matrix has been interchanged with row IPIV(i).

JPIV Input parameter.

JPIV is INTEGER array, dimension (N).

The pivot indices; for 1 <= j <= N, column j of the matrix has been interchanged with column JPIV(j).

Related Information

For this routine in other precisions, please see *clatdf*, *dlatdf* and *slatdf*.

4.17.772 zlatps

zlatps solves one of the triangular systems

```
A * x = s*b,  A**T * x = s*b,  or  A**H * x = s*b,
```

with scaling to prevent overflow, where A is an upper or lower triangular matrix stored in packed form. Here A^T denotes the transpose of A, A^H denotes the conjugate transpose of A, x and b are n-element vectors, and s is a scaling factor, usually less than or equal to 1, chosen so that the components of x will be less than the overflow threshold. If the unscaled problem will not cause overflow, the Level 2 BLAS routine ZTPSV is called. If the matrix A is singular (A(j,j) = 0 for some j), then s is set to 0 and a non-trivial solution to A*x = 0 is returned.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlatps(UPLO, TRANS, DIAG, NORMIN, N, AP, X, SCALE, CNORM, INFO)
```

C specification:

```
#include "armpl.h"

void zlatps_(const char *uplo, const char *trans, const char *diag,
             const char *normin, const armpl_int_t *n,
             const armpl_doublecomplex_t *ap, armpl_doublecomplex_t *x,
             double *scale, double *cnorm, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the matrix A is upper or lower triangular. = 'U': Upper triangular = 'L': Lower triangular

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the operation applied to A. = 'N': Solve A * x = s*b (No transpose) = 'T': Solve A^T * x = s*b (Transpose) = 'C': Solve A^H * x = s*b (Conjugate transpose)

DIAG Input parameter.

DIAG is CHARACTER*1

Specifies whether or not the matrix A is unit triangular. = 'N': Non-unit triangular = 'U': Unit triangular

NORMIN Input parameter.

NORMIN is CHARACTER*1

Specifies whether CNORM has been set or not. = 'Y': CNORM contains the column norms on entry = 'N': CNORM is not set on entry. On exit, the norms will be computed and stored in CNORM.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2). The upper or lower triangular matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension (N). On entry, the right hand side b of the triangular system. On exit, X is overwritten by the solution vector x.

SCALE Output parameter.

SCALE is DOUBLE PRECISION

The scaling factor s for the triangular system A * x = s*b, A^T * x = s*b, or A^H * x = s*b. If SCALE = 0, the matrix A is singular or badly scaled, and the vector x is an exact or approximate solution to A*x = 0.

CNORM Input and output parameter.

CNORM is DOUBLE PRECISION

CNORM is an array, dimension (N). If NORMIN = 'Y', CNORM is an input argument and CNORM(j) contains the norm of the off-diagonal part of the j-th column of A. If TRANS = 'N', CNORM(j) must be greater than or equal to the infinity-norm, and if TRANS = 'T' or 'C', CNORM(j) must be greater than or equal to the 1-norm.

If NORMIN = 'N', CNORM is an output argument and CNORM(j) returns the 1-norm of the offdiagonal part of the j-th column of A.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value

Related Information

For this routine in other precisions, please see *clatps*, *dlatps* and *slatps*.

4.17.773 zlatrd

`zlatrd` reduces NB rows and columns of a complex Hermitian matrix A to Hermitian tridiagonal form by a unitary similarity transformation Q^H * A * Q, and returns the matrices V and W which are needed to apply the transformation to the unreduced part of A.

If UPLO = 'U', `zlatrd` reduces the last NB rows and columns of a matrix, of which the upper triangle is supplied; if UPLO = 'L', `zlatrd` reduces the first NB rows and columns of a matrix, of which the lower triangle is supplied.

This is an auxiliary routine called by ZHETRD.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlatrd(UPLO, N, NB, A, LDA, E, TAU, W, LDW)
```

C specification:

```
#include "armpl.h"

void zlatrd_(const char *uplo, const armpl_int_t *n, const armpl_int_t *nb,
             armpl_doublecomplex_t *a, const armpl_int_t *lda, double *e,
             armpl_doublecomplex_t *tau, armpl_doublecomplex_t *w,
             const armpl_int_t *ldw, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the Hermitian matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A.

NB Input parameter.

NB is INTEGER

The number of rows and columns to be reduced.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. On exit: if UPLO = 'U', the last NB columns have been reduced to tridiagonal form, with the diagonal elements overwriting the diagonal elements of A; the elements above the diagonal with the array TAU, represent the unitary matrix Q as a

product of elementary reflectors; if UPLO = 'L', the first NB columns have been reduced to tridiagonal form, with the diagonal elements overwriting the diagonal elements of A; the elements below the diagonal with the array TAU, represent the unitary matrix Q as a product of elementary reflectors. See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

E Output parameter.

E is DOUBLE PRECISION

E is an array, dimension (N-1). If UPLO = 'U', E(n-nb:n-1) contains the superdiagonal elements of the last NB columns of the reduced matrix; if UPLO = 'L', E(1:nb) contains the subdiagonal elements of the first NB columns of the reduced matrix.

TAU Output parameter.

TAU is COMPLEX*16

TAU is an array, dimension (N-1). The scalar factors of the elementary reflectors, stored in TAU(n-nb:n-1) if UPLO = 'U', and in TAU(1:nb) if UPLO = 'L'. See Further Details.

W Output parameter.

W is COMPLEX*16

W is an array, dimension (LDW, NB). The n-by-nb matrix W required to update the unreduced part of A.

LDW Input parameter.

LDW is INTEGER

The leading dimension of the array W. LDW >= max(1, N).

Related Information

For this routine in other precisions, please see *clatrd*, *dlatrd* and *slatrd*.

4.17.774 zlatrs

zlatrs solves one of the triangular systems

```
A * x = s*b,  A**T * x = s*b,  or  A**H * x = s*b,
```

with scaling to prevent overflow. Here A is an upper or lower triangular matrix, A^T denotes the transpose of A, A^H denotes the conjugate transpose of A, x and b are n-element vectors, and s is a scaling factor, usually less than or equal to 1, chosen so that the components of x will be less than the overflow threshold. If the unscaled problem will not cause overflow, the Level 2 BLAS routine ZTRSV is called. If the matrix A is singular (A(j,j) = 0 for some j), then s is set to 0 and a non-trivial solution to A*x = 0 is returned.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlatrs(UPLO, TRANS, DIAG, NORMIN, N, A, LDA, X, SCALE, CNORM,
                  INFO)
```

C specification:

```
#include "armpl.h"

void zlatrs_(const char *uplo, const char *trans, const char *diag,
             const char *normin, const armpl_int_t *n,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_doublecomplex_t *x, double *scale, double *cnorm,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the matrix A is upper or lower triangular. = 'U': Upper triangular = 'L': Lower triangular

TRANS Input parameter.

TRANS is CHARACTER*1

Specifies the operation applied to A. = 'N': Solve A * x = s*b (No transpose) = 'T': Solve A^T * x = s*b (Transpose) = 'C': Solve A^H * x = s*b (Conjugate transpose)

DIAG Input parameter.

DIAG is CHARACTER*1

Specifies whether or not the matrix A is unit triangular. = 'N': Non-unit triangular = 'U': Unit triangular

NORMIN Input parameter.

NORMIN is CHARACTER*1

Specifies whether CNORM has been set or not. = 'Y': CNORM contains the column norms on entry = 'N': CNORM is not set on entry. On exit, the norms will be computed and stored in CNORM.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The triangular matrix A. If UPLO = 'U', the leading n by n upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n by n lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max (1, N).

X Input and output parameter.

X is COMPLEX*16

X is an array, dimension (N). On entry, the right hand side b of the triangular system. On exit, X is overwritten by the solution vector x.

SCALE Output parameter.

SCALE is DOUBLE PRECISION

The scaling factor s for the triangular system A * x = s*b, A^T * x = s*b, or A^H * x = s*b. If SCALE = 0, the matrix A is singular or badly scaled, and the vector x is an exact or approximate solution to A*x = 0.

CNORM Input and output parameter.

CNORM is DOUBLE PRECISION

CNORM is an array, dimension (N). If NORMIN = 'Y', CNORM is an input argument and CNORM(j) contains the norm of the off-diagonal part of the j-th column of A. If TRANS = 'N', CNORM(j) must be greater than or equal to the infinity-norm, and if TRANS = 'T' or 'C', CNORM(j) must be greater than or equal to the 1-norm.

If NORMIN = 'N', CNORM is an output argument and CNORM(j) returns the 1-norm of the offdiagonal part of the j-th column of A.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value

Related Information

For this routine in other precisions, please see *clatrs*, *dlatrs* and *slatrs*.

4.17.775 zlatrz

zlatrz factors the M-by-(M+L) complex upper trapezoidal matrix [A1 A2] = [A(1:M,1:M) A(1:M,N-L+1:N)] as (R 0) * Z by means of unitary transformations, where Z is an (M+L)-by-(M+L) unitary matrix and, R and A1 are M-by-M upper triangular matrices.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlatrz(M, N, L, A, LDA, TAU, WORK)
```

C specification:

```
#include "armpl.h"

void zlatrz_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *l,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_doublecomplex_t *tau, armpl_doublecomplex_t *work);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. N >= 0.

L Input parameter.

L is INTEGER

The number of columns of the matrix A containing the meaningful part of the Householder vectors. N-M >= L >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the leading M-by-N upper trapezoidal part of the array A must contain the matrix to be factorized. On exit, the leading M-by-M upper triangular part of A contains the upper triangular matrix R, and elements N-L+1 to N of the first M rows of A, with the array TAU, represent the unitary matrix Z as a product of M elementary reflectors.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

TAU Output parameter.

TAU is COMPLEX*16

TAU is an array, dimension (M). The scalar factors of the elementary reflectors.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (M) .

Related Information

For this routine in other precisions, please see *clatrz*, *dlatrz* and *slatrz*.

4.17.776 zlatsqr

SLATSQR computes a blocked Tall-Skinny QR factorization of an M-by-N matrix A, where M >= N: A = Q * R .

Syntax

Fortran specification:

```
use armpl_library

subroutine zlatsqr(M, N, MB, NB, A, LDA, T, LDT, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zlatsqr_(const armpl_int_t *m, const armpl_int_t *n,
              const armpl_int_t *mb, const armpl_int_t *nb,
              armpl_doublecomplex_t *a, const armpl_int_t *lda,
              armpl_doublecomplex_t *t, const armpl_int_t *ldt,
              armpl_doublecomplex_t *work, const armpl_int_t *lwork,
              armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix A. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix A. M >= N >= 0.

MB Input parameter.

MB is INTEGER

The row block size to be used in the blocked QR. MB > N.

NB Input parameter.

NB is INTEGER

The column block size to be used in the blocked QR. N >= NB >= 1.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the M-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the N-by-N upper triangular matrix R; the elements below the diagonal represent Q by the columns of blocked V (see Further Details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

T Output parameter.

T is COMPLEX*16

T is an array,. dimension (LDT, N * Number_of_row_blocks) where Number_of_row_blocks = CEIL((M-N)/(MB-N)) The blocked upper triangular block reflectors stored in compact form as a sequence of upper triangular blocks. See Further Details below.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= NB.

WORK Output parameter.

(workspace) COMPLEX*16

(workspace) is an array, dimension (MAX(1, LWORK)) .

LWORK Input parameter.

The dimension of the array WORK. LWORK >= NB*N.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *clatsqr*, *dlatsqr* and *slatsqr*.

4.17.777 zlauu2

`zlauu2` computes the product U * U^H or L^H * L, where the triangular factor U or L is stored in the upper or lower triangular part of the array A.

If UPLO = 'U' or 'u' then the upper triangle of the result is stored, overwriting the factor U in A. If UPLO = 'L' or 'l' then the lower triangle of the result is stored, overwriting the factor L in A.

This is the unblocked form of the algorithm, calling Level 2 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlauu2(UPLO, N, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void zlauu2_(const char *uplo, const armpl_int_t *n, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the triangular factor stored in the array A is upper or lower triangular: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the triangular factor U or L. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product U * U^H ; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product L^H * L.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value

Related Information

For this routine in other precisions, please see *clauu2*, *dlauu2* and *slauu2*.

4.17.778 zlauum

`zlauum` computes the product U * U^H or L^H * L, where the triangular factor U or L is stored in the upper or lower triangular part of the array A.

If UPLO = 'U' or 'u' then the upper triangle of the result is stored, overwriting the factor U in A. If UPLO = 'L' or 'l' then the lower triangle of the result is stored, overwriting the factor L in A.

This is the blocked form of the algorithm, calling Level 3 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine zlauum(UPLO, N, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void zlauum_(const char *uplo, const armpl_int_t *n, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the triangular factor stored in the array A is upper or lower triangular: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the triangular factor U or L. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product U * U^H ; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product L^H * L.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value

Related Information

For this routine in other precisions, please see *clauum*, *dlauum* and *slauum*. It also exists with a native C interface as *LAPACKE_zlauum*.

4.17.779 zpbtf2

`zpbtf2` computes the Cholesky factorization of a complex Hermitian positive definite band matrix A.

The factorization has the form

```
A = U**H * U ,  if UPLO = 'U', or
A = L  * L**H,  if UPLO = 'L',
```

where U is an upper triangular matrix, U^H is the conjugate transpose of U, and L is lower triangular.

This is the unblocked version of the algorithm, calling Level 2 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine zpbtf2(UPLO, N, KD, AB, LDAB, INFO)
```

C specification:

```
#include "armpl.h"

void zpbtf2_(const char *uplo, const armpl_int_t *n, const armpl_int_t *kd,
             armpl_doublecomplex_t *ab, const armpl_int_t *ldab,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

`UPLO` is `CHARACTER*1`

Specifies whether the upper or lower triangular part of the Hermitian matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

`N` is `INTEGER`

The order of the matrix A. `N` >= 0.

KD Input parameter.

`KD` is `INTEGER`

The number of super-diagonals of the matrix A if `UPLO` = 'U', or the number of sub-diagonals if `UPLO` = 'L'. `KD` >= 0.

AB Input and output parameter.

`AB` is `COMPLEX*16`

`AB` is an array, dimension (`LDAB`, `N`). On entry, the upper or lower triangle of the Hermitian band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB

as follows: if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j; if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

On exit, if INFO = 0, the triangular factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H of the band matrix A, in the same storage format as A.

LDAB Input parameter.

LDAB is INTEGER

The leading dimension of the array AB. LDAB >= KD+1.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value > 0: if INFO = k, the leading minor of order k is not positive definite, and the factorization could not be completed.

Related Information

For this routine in other precisions, please see *cpbtf2*, *dpbtf2* and *spbtf2*.

4.17.780 zpotf2

zpotf2 computes the Cholesky factorization of a complex Hermitian positive definite matrix A.

The factorization has the form

```
A = U**H * U ,  if UPLO = 'U', or
A = L  * L**H,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is lower triangular.

This is the unblocked version of the algorithm, calling Level 2 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine zpotf2(UPLO, N, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void zpotf2_(const char *uplo, const armpl_int_t *n, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the Hermitian matrix A is stored. = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the Hermitian matrix A. If UPLO = 'U', the leading n by n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n by n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization A = U^H *U or A = L*L^H .

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value > 0: if INFO = k, the leading minor of order k is not positive definite, and the factorization could not be completed.

Related Information

For this routine in other precisions, please see *cpotf2*, *dpotf2* and *spotf2*.

4.17.781 zpstf2

zpstf2 computes the Cholesky factorization with complete pivoting of a complex Hermitian positive semidefinite matrix A.

The factorization has the form

```
P**T * A * P = U**H * U ,  if UPLO = 'U',
P**T * A * P = L  * L**H,  if UPLO = 'L',
```

where U is an upper triangular matrix and L is lower triangular, and P is stored as vector PIV.

This algorithm does not attempt to check that A is positive semidefinite. This version of the algorithm calls level 2 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine zpstf2(UPLO, N, A, LDA, PIV, RANK, TOL, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zpstf2_(const char *uplo, const armpl_int_t *n, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *piv, armpl_int_t *rank,
             const double *tol, double *work, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored. = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading n by n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n by n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization as above.

PIV Output parameter.

PIV is INTEGER array, dimension (N)

PIV is such that the nonzero entries are P(PIV(K), K) = 1.

RANK Output parameter.

RANK is INTEGER

The rank of A given by the number of steps the algorithm completed.

TOL Input parameter.

TOL is DOUBLE PRECISION

User defined tolerance. If TOL < 0, then N*U*MAX(A(K,K)) will be used. The algorithm terminates at the (K-1)st step if the pivot <= TOL.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

WORK Output parameter.

WORK is DOUBLE PRECISION

WORK is an array, dimension (2*N). Work space.

INFO Output parameter.

INFO is INTEGER

< 0: If INFO = -K, the K-th argument had an illegal value, = 0: algorithm completed successfully, and > 0: the matrix A is either rank deficient with computed rank as returned in RANK, or is not positive semidefinite. See Section 7 of LAPACK Working Note #161 for further information.

Related Information

For this routine in other precisions, please see *cpstf2*, *dpstf2* and *spstf2*.

4.17.782 zptts2

zptts2 solves a tridiagonal system of the form

```
A * X = B
```

using the factorization A = U^H *D*U or A = L*D*L^H computed by ZPTTRF. D is a diagonal matrix specified in the vector D, U (or L) is a unit bidiagonal matrix whose superdiagonal (subdiagonal) is specified in the vector E, and X and B are N by NRHS matrices.

Syntax

Fortran specification:

```
use armpl_library

subroutine zptts2(IUPLO, N, NRHS, D, E, B, LDB)
```

C specification:

```
#include "armpl.h"

void zptts2_(const armpl_int_t *iuplo, const armpl_int_t *n,
             const armpl_int_t *nrhs, const double *d,
             const armpl_doublecomplex_t *e, armpl_doublecomplex_t *b,
             const armpl_int_t *ldb);
```

Parameters

IUPLO Input parameter.

IUPLO is INTEGER

Specifies the form of the factorization and whether the vector E is the superdiagonal of the upper bidiagonal factor U or the subdiagonal of the lower bidiagonal factor L. = 1: A = U^H *D*U, E is the superdiagonal of U = 0: A = L*D*L^H , E is the subdiagonal of L

N Input parameter.

N is INTEGER

The order of the tridiagonal matrix A. N >= 0.

NRHS Input parameter.

NRHS is INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

D Input parameter.

D is DOUBLE PRECISION

D is an array, dimension (N). The n diagonal elements of the diagonal matrix D from the factorization A = U^H *D*U or A = L*D*L^H .

E Input parameter.

E is COMPLEX*16

E is an array, dimension (N-1). If IUPLO = 1, the (n-1) superdiagonal elements of the unit bidiagonal factor U from the factorization A = U^H *D*U. If IUPLO = 0, the (n-1) subdiagonal elements of the unit bidiagonal factor L from the factorization A = L*D*L^H .

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, NRHS). On entry, the right hand side vectors B for the system of linear equations. On exit, the solution vectors, X.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Related Information

For this routine in other precisions, please see *cptts2*, *dptts2* and *sptts2*.

4.17.783 zsyconv

zsyconv converts A given by ZHETRF into L and D or vice-versa. Get nondiagonal elements of D (returned in workspace) and apply or reverse permutation done in TRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsyconv(UPLO, WAY, N, A, LDA, IPIV, E, INFO)
```

C specification:

```
#include "armpl.h"

void zsyconv_(const char *uplo, const char *way, const armpl_int_t *n,
              armpl_doublecomplex_t *a, const armpl_int_t *lda,
              const armpl_int_t *ipiv, armpl_doublecomplex_t *e,
              armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

WAY Input parameter.

WAY is CHARACTER*1

= 'C': Convert = 'R': Revert

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by ZSYTRF.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Input parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D as determined by ZSYTRF.

E Output parameter.

E is COMPLEX*16

E is an array, dimension (N). E stores the supdiagonal/subdiagonal of the symmetric 1-by-1 or 2-by-2 block diagonal matrix D in LDLT.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *csyconv*, *dsyconv* and *ssyconv*. It also exists with a native C interface as *LAPACKE_zsyconv*.

4.17.784 zsyswapr

ZSYSWAPR applies an elementary permutation on the rows and the columns of a symmetric matrix.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsyswapr(UPLO, N, A, LDA, I1, I2)
```

C specification:

```
#include "armpl.h"

void zsyswapr_(const char *uplo, const armpl_int_t *n,
               armpl_doublecomplex_t *a, const armpl_int_t *lda,
               const armpl_int_t *i1, const armpl_int_t *i2, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the details of the factorization are stored as an upper or lower triangular matrix. = 'U': Upper triangular, form is A = U*D*U^T ; = 'L': Lower triangular, form is A = L*D*L^T .

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the NB diagonal matrix D and the multipliers used to obtain the factor U or L as computed by ZSYTRF.

On exit, if INFO = 0, the (symmetric) inverse of the original matrix. If UPLO = 'U', the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; if UPLO = 'L' the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

I1 Input parameter.

I1 is INTEGER

Index of the first row to swap

I2 Input parameter.

I2 is INTEGER

Index of the second row to swap

Related Information

For this routine in other precisions, please see *csyswapr*, *dsyswapr* and *ssyswapr*. It also exists with a native C interface as *LAPACKE_zsyswapr*.

4.17.785 zsytf2

zsytf2 computes the factorization of a complex symmetric matrix A using the Bunch-Kaufman diagonal pivoting method:

```
A = U*D*U**T  or  A = L*D*L**T
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, U^T is the transpose of U, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the unblocked version of the algorithm, calling Level 2 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsytf2(UPLO, N, A, LDA, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void zsytf2_(const char *uplo, const armpl_int_t *n, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *ipiv, armpl_int_t *info,
             ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L (see below for further details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D.

If UPLO = 'U': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) = IPIV(k-1) < 0, then rows and columns k-1 and -IPIV(k) were interchanged and D(k-1:k,k-1:k) is a 2-by-2 diagonal block.

If UPLO = 'L': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and -IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value > 0: if INFO = k, D(k,k) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *csytf2*, *dsytf2* and *ssytf2*.

4.17.786 zsytf2_rook

ZSYTF2_ROOK computes the factorization of a complex symmetric matrix A using the bounded Bunch-Kaufman ("rook") diagonal pivoting method:

```
A = U*D*U**T  or  A = L*D*L**T
```

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, U^T is the transpose of U, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

This is the unblocked version of the algorithm, calling Level 2 BLAS.

Syntax

Fortran specification:

```
use armpl_library

subroutine zsytf2_rook(UPLO, N, A, LDA, IPIV, INFO)
```

C specification:

```
#include "armpl.h"

void zsytf2_rook_(const char *uplo, const armpl_int_t *n,
                  armpl_doublecomplex_t *a, const armpl_int_t *lda,
                  armpl_int_t *ipiv, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the upper or lower triangular part of the symmetric matrix A is stored: = 'U': Upper triangular = 'L': Lower triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the symmetric matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L (see below for further details).

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

IPIV Output parameter.

IPIV is INTEGER array, dimension (N)

Details of the interchanges and the block structure of D.

If UPLO = 'U': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k-1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k-1 and -IPIV(k-1) were inerchaged, D(k-1:k,k-1:k) is a 2-by-2 diagonal block.

If UPLO = 'L': If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.

If IPIV(k) < 0 and IPIV(k+1) < 0, then rows and columns k and -IPIV(k) were interchanged and rows and columns k+1 and -IPIV(k+1) were inerchaged, D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit

< 0: if INFO = -k, the k-th argument had an illegal value

> 0: if INFO = k, D(k,k) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Related Information

For this routine in other precisions, please see *csytf2_rook*, *dsytf2_rook* and *ssytf2_rook*.

4.17.787 ztfsm

Level 3 BLAS like routine for A in RFP Format.

ztfsm solves the matrix equation

```
op( A )*X = alpha*B  or  X*op( A ) = alpha*B
```

where alpha is a scalar, X and B are m by n matrices, A is a unit, or non-unit, upper or lower triangular matrix and op(A) is one of

```
op( A ) = A   or   op( A ) = A**H.
```

A is in Rectangular Full Packed (RFP) Format.

The matrix X is overwritten on B.

Syntax

Fortran specification:

```
use armpl_library

subroutine ztfsm(TRANSR, SIDE, UPLO, TRANS, DIAG, M, N, ALPHA, A, B, LDB)
```

C specification:

```
#include "armpl.h"

void ztfsm_(const char *transr, const char *side, const char *uplo,
            const char *trans, const char *diag, const armpl_int_t *m,
            const armpl_int_t *n, const armpl_doublecomplex_t *alpha,
            const armpl_doublecomplex_t *a, armpl_doublecomplex_t *b,
            const armpl_int_t *ldb, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': The Normal Form of RFP A is stored; = 'C': The Conjugate-transpose Form of RFP A is stored.

SIDE Input parameter.

SIDE is CHARACTER*1

On entry, SIDE specifies whether op(A) appears on the left or right of X as follows:

SIDE = 'L' or 'l' op(A)*X = alpha*B.

SIDE = 'R' or 'r' X*op(A) = alpha*B.

Unchanged on exit.

UPLO Input parameter.

UPLO is CHARACTER*1

On entry, UPLO specifies whether the RFP matrix A came from an upper or lower triangular matrix as follows: UPLO = 'U' or 'u' RFP A came from an upper triangular matrix UPLO = 'L' or 'l' RFP A came from a lower triangular matrix

Unchanged on exit.

TRANS Input parameter.

TRANS is CHARACTER*1

On entry, TRANS specifies the form of op(A) to be used in the matrix multiplication as follows:

TRANS = 'N' or 'n' op(A) = A.

TRANS = 'C' or 'c' op(A) = conjg(A').

Unchanged on exit.

DIAG Input parameter.

DIAG is CHARACTER*1

On entry, DIAG specifies whether or not RFP A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

Unchanged on exit.

M Input parameter.

M is INTEGER

On entry, M specifies the number of rows of B. M must be at least zero. Unchanged on exit.

N Input parameter.

N is INTEGER

On entry, N specifies the number of columns of B. N must be at least zero. Unchanged on exit.

ALPHA Input parameter.

ALPHA is COMPLEX*16

On entry, ALPHA specifies the scalar alpha. When alpha is zero then A is not referenced and B need not be set before entry. Unchanged on exit.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (N*(N+1)/2). NT = N*(N+1)/2. On entry, the matrix A in RFP Format. RFP Format is described by TRANSR, UPLO and N as follows: If TRANSR='N' then RFP A is (0:N,0:K-1) when N is even; K=N/2. RFP A is (0:N-1,0:K) when N is odd; K=N/2. If TRANSR = 'C' then RFP is the Conjugate-transpose of RFP A as defined when TRANSR = 'N'. The contents of RFP A are defined by UPLO as follows: If UPLO = 'U' the RFP A contains the NT elements of upper packed A either in normal or conjugate-transpose Format. If UPLO = 'L' the RFP A contains the NT elements of lower packed A either in normal or conjugate-transpose Format. The LDA of RFP A is (N+1)/2 when TRANSR = 'C'. When TRANSR is 'N' the LDA is N+1 when N is even and is N when is odd. See the Note below for more details. Unchanged on exit.

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). Before entry, the leading m by n part of the array B must contain the right-hand side matrix B, and on exit is overwritten by the solution matrix X.

LDB Input parameter.

LDB is INTEGER

On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. LDB must be at least max(1, m). Unchanged on exit.

Related Information

For this routine in other precisions, please see *ctfsm*, *dtfsm* and *stfsm*. It also exists with a native C interface as *LAPACKE_ztfsm*.

4.17.788 ztfttp

ztfttp copies a triangular matrix A from rectangular full packed format (TF) to standard packed format (TP).

Syntax

Fortran specification:

```
use armpl_library

subroutine ztfttp(TRANSR, UPLO, N, ARF, AP, INFO)
```

C specification:

```
#include "armpl.h"

void ztfttp_(const char *transr, const char *uplo, const armpl_int_t *n,
             const armpl_doublecomplex_t *arf, armpl_doublecomplex_t *ap,
             armpl_int_t *info, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': ARF is in Normal format; = 'C': ARF is in Conjugate-transpose format;

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

ARF Input parameter.

ARF is COMPLEX*16

ARF is an array, dimension (N*(N+1)/2),. On entry, the upper or lower triangular matrix A stored in RFP format. For a further discussion see Notes below.

AP Output parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2),. On exit, the upper or lower triangular matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctfttp*, *dtfttp* and *stfttp*. It also exists with a native C interface as *LAPACKE_ztfttp*.

4.17.789 ztfttr

ztfttr copies a triangular matrix A from rectangular full packed format (TF) to standard full format (TR).

Syntax

Fortran specification:

```
use armpl_library

subroutine ztfttr(TRANSR, UPLO, N, ARF, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void ztfttr_(const char *transr, const char *uplo, const armpl_int_t *n,
             const armpl_doublecomplex_t *arf, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *info, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': ARF is in Normal format; = 'C': ARF is in Conjugate-transpose format;

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

ARF Input parameter.

ARF is COMPLEX*16

ARF is an array, dimension (N*(N+1)/2),. On entry, the upper or lower triangular matrix A stored in RFP format. For a further discussion see Notes below.

A Output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On exit, the triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctfttr*, *dtfttr* and *stfttr*. It also exists with a native C interface as *LAPACKE_ztfttr*.

4.17.790 ztgex2

ztgex2 swaps adjacent diagonal 1 by 1 blocks (A11,B11) and (A22,B22) in an upper triangular matrix pair (A, B) by an unitary equivalence transformation.

(A, B) must be in generalized Schur canonical form, that is, A and B are both upper triangular.

Optionally, the matrices Q and Z of generalized Schur vectors are updated.

```
Q(in) * A(in) * Z(in)**H = Q(out) * A(out) * Z(out)**H
Q(in) * B(in) * Z(in)**H = Q(out) * B(out) * Z(out)**H
```

Syntax

Fortran specification:

```
use armpl_library

subroutine ztgex2(WANTQ, WANTZ, N, A, LDA, B, LDB, Q, LDQ, Z, LDZ, J1, INFO)
```

C specification:

```
#include "armpl.h"

void ztgex2_(const armpl_int_t *wantq, const armpl_int_t *wantz,
             const armpl_int_t *n, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, armpl_doublecomplex_t *q,
             const armpl_int_t *ldq, armpl_doublecomplex_t *z,
             const armpl_int_t *ldz, const armpl_int_t *j1,
             armpl_int_t *info);
```

Parameters

WANTQ Input parameter.

WANTQ is LOGICAL

.TRUE. : update the left transformation matrix Q; .FALSE.: do not update Q.

WANTZ Input parameter.

WANTZ is LOGICAL

.TRUE. : update the right transformation matrix Z; .FALSE.: do not update Z.

N Input parameter.

N is INTEGER

The order of the matrices A and B. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimensions (LDA, N). On entry, the matrix A in the pair (A, B). On exit, the updated matrix A.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimensions (LDB, N). On entry, the matrix B in the pair (A, B). On exit, the updated matrix B.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, N).

Q Input and output parameter.

Q is COMPLEX*16

Q is an array, dimension (LDQ, N). If WANTQ = .TRUE, on entry, the unitary matrix Q. On exit, the updated matrix Q. Not referenced if WANTQ = .FALSE..

LDQ Input parameter.

LDQ is INTEGER

The leading dimension of the array Q. LDQ >= 1; If WANTQ = .TRUE., LDQ >= N.

Z Input and output parameter.

Z is COMPLEX*16

Z is an array, dimension (LDZ, N). If WANTZ = .TRUE, on entry, the unitary matrix Z. On exit, the updated matrix Z. Not referenced if WANTZ = .FALSE..

LDZ Input parameter.

LDZ is INTEGER

The leading dimension of the array Z. LDZ >= 1; If WANTZ = .TRUE., LDZ >= N.

J1 Input parameter.

J1 is INTEGER

The index to the first block (A11, B11).

INFO Output parameter.

INFO is INTEGER

=0: Successful exit. =1: The transformed matrix pair (A, B) would be too far from generalized Schur form; the problem is ill- conditioned.

Related Information

For this routine in other precisions, please see *ctgex2*, *dtgex2* and *stgex2*.

4.17.791 ztgsy2

ztgsy2 solves the generalized Sylvester equation

```
A * R - L * B = scale * C                (1)
D * R - L * E = scale * F
```

using Level 1 and 2 BLAS, where R and L are unknown M-by-N matrices, (A, D), (B, E) and (C, F) are given matrix pairs of size M-by-M, N-by-N and M-by-N, respectively. A, B, D and E are upper triangular (i.e., (A,D) and (B,E) in generalized Schur form).

The solution (R, L) overwrites (C, F). 0 <= SCALE <= 1 is an output scaling factor chosen to avoid overflow.

In matrix notation solving equation (1) corresponds to solve Zx = scale * b, where Z is defined as

```
Z = [ kron(In, A)  -kron(B**H, Im) ]             (2)
    [ kron(In, D)  -kron(E**H, Im) ],
```

Ik is the identity matrix of size k and X^H is the conjuguate transpose of X. kron(X, Y) is the Kronecker product between the matrices X and Y.

If TRANS = 'C', y in the conjugate transposed system Z^H *y = scale*b is solved for, which is equivalent to solve for R and L in

```
A**H * R  + D**H * L   = scale * C           (3)
R  * B**H + L  * E**H  = scale * -F
```

This case is used to compute an estimate of Dif[(A, D), (B, E)] = = sigma_min(Z) using reverse communicaton with ZLACON.

ztgsy2 also (IJOB >= 1) contributes to the computation in ZTGSYL of an upper bound on the separation between to matrix pairs. Then the input (A, D), (B, E) are sub-pencils of two matrix pairs in ZTGSYL.

Syntax

Fortran specification:

```
use armpl_library

subroutine ztgsy2(TRANS, IJOB, M, N, A, LDA, B, LDB, C, LDC, D, LDD, E, LDE,
                  F, LDF, SCALE, RDSUM, RDSCAL, INFO)
```

C specification:

```
#include "armpl.h"

void ztgsy2_(const char *trans, const armpl_int_t *ijob, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_doublecomplex_t *a,
             const armpl_int_t *lda, const armpl_doublecomplex_t *b,
             const armpl_int_t *ldb, armpl_doublecomplex_t *c,
             const armpl_int_t *ldc, const armpl_doublecomplex_t *d,
             const armpl_int_t *ldd, const armpl_doublecomplex_t *e,
             const armpl_int_t *lde, armpl_doublecomplex_t *f,
             const armpl_int_t *ldf, double *scale, double *rdsum,
             double *rdscal, armpl_int_t *info, ... );
```

Parameters

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N', solve the generalized Sylvester equation (1). = 'T': solve the 'transposed' system (3).

IJOB Input parameter.

IJOB is INTEGER

Specifies what kind of functionality to be performed. =0: solve (1) only. =1: A contribution from this subsystem to a Frobenius norm-based estimate of the separation between two matrix pairs is computed. (look ahead strategy is used). =2: A contribution from this subsystem to a Frobenius norm-based estimate of the separation between two matrix pairs is computed. (DGECON on sub-systems is used.) Not referenced if TRANS = 'T'.

M Input parameter.

M is INTEGER

On entry, M specifies the order of A and D, and the row dimension of C, F, R and L.

N Input parameter.

N is INTEGER

On entry, N specifies the order of B and E, and the column dimension of C, F, R and L.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, M). On entry, A contains an upper triangular matrix.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the matrix A. LDA >= max(1, M).

B Input parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). On entry, B contains an upper triangular matrix.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the matrix B. LDB >= max(1, N).

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). On entry, C contains the right-hand-side of the first matrix equation in (1). On exit, if IJOB = 0, C has been overwritten by the solution R.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the matrix C. LDC >= max(1, M).

D Input parameter.

D is COMPLEX*16

D is an array, dimension (LDD, M). On entry, D contains an upper triangular matrix.

LDD Input parameter.

LDD is INTEGER

The leading dimension of the matrix D. LDD >= max(1, M).

E Input parameter.

E is COMPLEX*16

E is an array, dimension (LDE, N). On entry, E contains an upper triangular matrix.

LDE Input parameter.

LDE is INTEGER

The leading dimension of the matrix E. LDE >= max(1, N).

F Input and output parameter.

F is COMPLEX*16

F is an array, dimension (LDF, N). On entry, F contains the right-hand-side of the second matrix equation in (1). On exit, if IJOB = 0, F has been overwritten by the solution L.

LDF Input parameter.

LDF is INTEGER

The leading dimension of the matrix F. LDF >= max(1, M).

SCALE Output parameter.

SCALE is DOUBLE PRECISION

On exit, 0 <= SCALE <= 1. If 0 < SCALE < 1, the solutions R and L (C and F on entry) will hold the solutions to a slightly perturbed system but the input matrices A, B, D and E have not been changed. If SCALE = 0, R and L will hold the solutions to the homogeneous system with C = F = 0. Normally, SCALE = 1.

RDSUM Input and output parameter.

RDSUM is DOUBLE PRECISION

On entry, the sum of squares of computed contributions to the Dif-estimate under computation by ZTGSYL, where the scaling factor RDSCAL (see below) has been factored out. On exit, the corresponding sum of squares updated with the contributions from the current sub-system. If TRANS = 'T' RDSUM is not touched. NOTE: RDSUM only makes sense when ZTGSY2 is called by ZTGSYL.

RDSCAL Input and output parameter.

RDSCAL is DOUBLE PRECISION

On entry, scaling factor used to prevent overflow in RDSUM. On exit, RDSCAL is updated w.r.t. the current contributions in RDSUM. If TRANS = 'T', RDSCAL is not touched. NOTE: RDSCAL only makes sense when ZTGSY2 is called by ZTGSYL.

INFO Output parameter.

INFO is INTEGER

On exit, if INFO is set to =0: Successful exit <0: If INFO = -i, input argument number i is illegal. >0: The matrix pairs (A, D) and (B, E) have common or very close eigenvalues.

Related Information

For this routine in other precisions, please see *ctgsy2*, *dtgsy2* and *stgsy2*.

4.17.792 ztplqt2

ztplqt2 computes a LQ a factorization of a complex "triangular-pentagonal" matrix C, which is composed of a triangular block A and pentagonal block B, using the compact WY representation for Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine ztplqt2(M, N, L, A, LDA, B, LDB, T, LDT, INFO)
```

C specification:

```
#include "armpl.h"

void ztplqt2_(const armpl_int_t *m, const armpl_int_t *n,
              const armpl_int_t *l, armpl_doublecomplex_t *a,
              const armpl_int_t *lda, armpl_doublecomplex_t *b,
              const armpl_int_t *ldb, armpl_doublecomplex_t *t,
              const armpl_int_t *ldt, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The total number of rows of the matrix B. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix B, and the order of the triangular matrix A. N >= 0.

L Input parameter.

L is INTEGER

The number of rows of the lower trapezoidal part of B. MIN(M, N) >= L >= 0. See Further Details.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, M). On entry, the lower triangular M-by-M matrix A. On exit, the elements on and below the diagonal of the array contain the lower triangular matrix L.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, M).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). On entry, the pentagonal M-by-N matrix B. The first N-L columns are rectangular, and the last L columns are lower trapezoidal. On exit, B contains the pentagonal matrix V. See Further Details.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M).

T Output parameter.

T is COMPLEX*16

T is an array, dimension (LDT, M). The N-by-N upper triangular factor T of the block reflector. See Further Details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, M)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctplqt2*, *dtplqt2* and *stplqt2*.

4.17.793 ztpqrt2

ztpqrt2 computes a QR factorization of a complex "triangular-pentagonal" matrix C, which is composed of a triangular block A and pentagonal block B, using the compact WY representation for Q.

Syntax

Fortran specification:

```
use armpl_library

subroutine ztpqrt2(M, N, L, A, LDA, B, LDB, T, LDT, INFO)
```

C specification:

```
#include "armpl.h"

void ztpqrt2_(const armpl_int_t *m, const armpl_int_t *n,
              const armpl_int_t *l, armpl_doublecomplex_t *a,
              const armpl_int_t *lda, armpl_doublecomplex_t *b,
              const armpl_int_t *ldb, armpl_doublecomplex_t *t,
              const armpl_int_t *ldt, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The total number of rows of the matrix B. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix B, and the order of the triangular matrix A. N >= 0.

L Input parameter.

L is INTEGER

The number of rows of the upper trapezoidal part of B. MIN(M, N) >= L >= 0. See Further Details.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the upper triangular N-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the upper triangular matrix R.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). On entry, the pentagonal M-by-N matrix B. The first M-L rows are rectangular, and the last L rows are upper trapezoidal. On exit, B contains the pentagonal matrix V. See Further Details.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M).

T Output parameter.

T is COMPLEX*16

T is an array, dimension (LDT, N). The N-by-N upper triangular factor T of the block reflector. See Further Details.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= max(1, N)

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctpqrt2*, *dtpqrt2* and *stpqrt2*. It also exists with a native C interface as *LAPACKE_ztpqrt2*.

4.17.794 ztprfb

ztprfb applies a complex "triangular-pentagonal" block reflector H or its conjugate transpose H^H to a complex matrix C, which is composed of two blocks A and B, either from the left or right.

Syntax

Fortran specification:

```
use armpl_library

subroutine ztprfb(SIDE, TRANS, DIRECT, STOREV, M, N, K, L, V, LDV, T, LDT, A,
                  LDA, B, LDB, WORK, LDWORK)
```

C specification:

```
#include "armpl.h"

void ztprfb_(const char *side, const char *trans, const char *direct,
             const char *storev, const armpl_int_t *m, const armpl_int_t *n,
             const armpl_int_t *k, const armpl_int_t *l,
             const armpl_doublecomplex_t *v, const armpl_int_t *ldv,
             const armpl_doublecomplex_t *t, const armpl_int_t *ldt,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_doublecomplex_t *b, const armpl_int_t *ldb,
             armpl_doublecomplex_t *work, const armpl_int_t *ldwork, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply H or H^H from the Left = 'R': apply H or H^H from the Right

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply H (No transpose) = 'C': apply H^H (Conjugate transpose)

DIRECT Input parameter.

DIRECT is CHARACTER*1

Indicates how H is formed from a product of elementary reflectors = 'F': H = H(1) H(2) . . . H(k) (Forward) = 'B': H = H(k) . . . H(2) H(1) (Backward)

STOREV Input parameter.

STOREV is CHARACTER*1

Indicates how the vectors which define the elementary reflectors are stored: = 'C': Columns = 'R': Rows

M Input parameter.

M is INTEGER

The number of rows of the matrix B. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix B. N >= 0.

K Input parameter.

K is INTEGER

The order of the matrix T, i.e. the number of elementary reflectors whose product defines the block reflector. K >= 0.

L Input parameter.

L is INTEGER

The order of the trapezoidal part of V. K >= L >= 0. See Further Details.

V Input parameter.

V is COMPLEX*16

V is an array, dimension. (LDV, K) if STOREV = 'C' (LDV, M) if STOREV = 'R' and SIDE = 'L' (LDV, N) if STOREV = 'R' and SIDE = 'R' The pentagonal matrix V, which contains the elementary reflectors H(1), H(2), . . . , H(K). See Further Details.

LDV Input parameter.

LDV is INTEGER

The leading dimension of the array V. If STOREV = 'C' and SIDE = 'L', LDV >= max(1, M); if STOREV = 'C' and SIDE = 'R', LDV >= max(1, N); if STOREV = 'R', LDV >= K.

T Input parameter.

T is COMPLEX*16

T is an array, dimension (LDT, K). The triangular K-by-K matrix T in the representation of the block reflector.

LDT Input parameter.

LDT is INTEGER

The leading dimension of the array T. LDT >= K.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension. (LDA, N) if SIDE = 'L' or (LDA, K) if SIDE = 'R' On entry, the K-by-N or M-by-K matrix A. On exit, A is overwritten by the corresponding block of H*C or H^H *C or C*H or C*H^H . See Further Details.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDC >= max(1, K); If SIDE = 'R', LDC >= max(1, M).

B Input and output parameter.

B is COMPLEX*16

B is an array, dimension (LDB, N). On entry, the M-by-N matrix B. On exit, B is overwritten by the corresponding block of H*C or H^H *C or C*H or C*H^H . See Further Details.

LDB Input parameter.

LDB is INTEGER

The leading dimension of the array B. LDB >= max(1, M).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension. (LDWORK, N) if SIDE = 'L', (LDWORK, K) if SIDE = 'R'.

LDWORK Input parameter.

LDWORK is INTEGER

The leading dimension of the array WORK. If SIDE = 'L', LDWORK >= K; if SIDE = 'R', LDWORK >= M.

Related Information

For this routine in other precisions, please see *ctprfb*, *dtprfb* and *stprfb*. It also exists with a native C interface as *LAPACKE_ztprfb*.

4.17.795 ztpttf

ztpttf copies a triangular matrix A from standard packed format (TP) to rectangular full packed format (TF).

Syntax

Fortran specification:

```
use armpl_library

subroutine ztpttf(TRANSR, UPLO, N, AP, ARF, INFO)
```

C specification:

```
#include "armpl.h"

void ztpttf_(const char *transr, const char *uplo, const armpl_int_t *n,
             const armpl_doublecomplex_t *ap, armpl_doublecomplex_t *arf,
             armpl_int_t *info, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': ARF in Normal format is wanted; = 'C': ARF in Conjugate-transpose format is wanted.

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2),. On entry, the upper or lower triangular matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

ARF Output parameter.

ARF is COMPLEX*16

ARF is an array, dimension (N*(N+1)/2),. On exit, the upper or lower triangular matrix A stored in RFP format. For a further discussion see Notes below.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctpttf*, *dtpttf* and *stpttf*. It also exists with a native C interface as *LAPACKE_ztpttf*.

4.17.796 ztpttr

ztpttr copies a triangular matrix A from standard packed format (TP) to standard full format (TR).

Syntax

Fortran specification:

```
use armpl_library

subroutine ztpttr(UPLO, N, AP, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void ztpttr_(const char *uplo, const armpl_int_t *n,
             const armpl_doublecomplex_t *ap, armpl_doublecomplex_t *a,
             const armpl_int_t *lda, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular. = 'L': A is lower triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

AP Input parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2),. On entry, the upper or lower triangular matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

A Output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On exit, the triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctpttr*, *dtpttr* and *stpttr*. It also exists with a native C interface as *LAPACKE_ztpttr*.

4.17.797 ztrti2

ztrti2 computes the inverse of a complex upper or lower triangular matrix.

This is the Level 2 BLAS version of the algorithm.

Syntax

Fortran specification:

```
use armpl_library

subroutine ztrti2(UPLO, DIAG, N, A, LDA, INFO)
```

C specification:

```
#include "armpl.h"

void ztrti2_(const char *uplo, const char *diag, const armpl_int_t *n,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

Specifies whether the matrix A is upper or lower triangular. = 'U': Upper triangular = 'L': Lower triangular

DIAG Input parameter.

DIAG is CHARACTER*1

Specifies whether or not the matrix A is unit triangular. = 'N': Non-unit triangular = 'U': Unit triangular

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the triangular matrix A. If UPLO = 'U', the leading n by n upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n by n lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1.

On exit, the (triangular) inverse of the original matrix, in the same storage format.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -k, the k-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctrti2*, *dtrti2* and *strti2*.

4.17.798 ztrttf

ztrttf copies a triangular matrix A from standard full format (TR) to rectangular full packed format (TF) .

Syntax

Fortran specification:

```
use armpl_library

subroutine ztrttf(TRANSR, UPLO, N, A, LDA, ARF, INFO)
```

C specification:

```
#include "armpl.h"

void ztrttf_(const char *transr, const char *uplo, const armpl_int_t *n,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_doublecomplex_t *arf, armpl_int_t *info, ... );
```

Parameters

TRANSR Input parameter.

TRANSR is CHARACTER*1

= 'N': ARF in Normal mode is wanted; = 'C': ARF in Conjugate Transpose mode is wanted;

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

N Input parameter.

N is INTEGER

The order of the matrix A. N >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the matrix A. LDA >= max(1, N).

ARF Output parameter.

ARF is COMPLEX*16

ARF is an array, dimension (N*(N+1)/2),. On exit, the upper or lower triangular matrix A stored in RFP format. For a further discussion see Notes below.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctrttf*, *dtrttf* and *strttf*. It also exists with a native C interface as *LAPACKE_ztrttf*.

4.17.799 ztrttp

`ztrttp` copies a triangular matrix A from full format (TR) to standard packed format (TP).

Syntax

Fortran specification:

```
use armpl_library

subroutine ztrttp(UPLO, N, A, LDA, AP, INFO)
```

C specification:

```
#include "armpl.h"

void ztrttp_(const char *uplo, const armpl_int_t *n,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             armpl_doublecomplex_t *ap, armpl_int_t *info, ... );
```

Parameters

UPLO Input parameter.

UPLO is CHARACTER*1

= 'U': A is upper triangular; = 'L': A is lower triangular.

N Input parameter.

N is INTEGER

The order of the matrices AP and A. N >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, N).

AP Output parameter.

AP is COMPLEX*16

AP is an array, dimension (N*(N+1)/2),. On exit, the upper or lower triangular matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array AP as follows: if UPLO = 'U', AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j; if UPLO = 'L', AP(i + (j-1)*(2n-j)/2) = A(i,j) for j<=i<=n.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *ctrttp*, *dtrttp* and *strttp*. It also exists with a native C interface as *LAPACKE_ztrttp*.

4.17.800 zunbdb1

`zunbdb1` simultaneously bidiagonalizes the blocks of a tall and skinny matrix X with orthonomal columns:

```
                        [ B11 ]
[ X11 ]   [ P1 |    ] [  0  ]
[-----] = [---------] [-----] Q1**T .
[ X21 ]   [    | P2 ] [ B21 ]
                        [  0  ]
```

X11 is P-by-Q, and X21 is (M-P)-by-Q. Q must be no larger than P, M-P, or M-Q. Routines ZUNBDB2, ZUNBDB3, and ZUNBDB4 handle cases in which Q is not the minimum dimension.

The unitary matrices P1, P2, and Q1 are P-by-P, (M-P)-by-(M-P), and (M-Q)-by-(M-Q), respectively. They are represented implicitly by Householder vectors.

B11 and B12 are Q-by-Q bidiagonal matrices represented implicitly by angles THETA, PHI.

Syntax

Fortran specification:

```
use armpl_library

subroutine zunbdb1(M, P, Q, X11, LDX11, X21, LDX21, THETA, PHI, TAUP1, TAUP2,
                   TAUQ1, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zunbdb1_(const armpl_int_t *m, const armpl_int_t *p,
              const armpl_int_t *q, armpl_doublecomplex_t *x11,
              const armpl_int_t *ldx11, armpl_doublecomplex_t *x21,
              const armpl_int_t *ldx21, double *theta, double *phi,
              armpl_doublecomplex_t *taup1, armpl_doublecomplex_t *taup2,
              armpl_doublecomplex_t *tauq1, armpl_doublecomplex_t *work,
              const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows X11 plus the number of rows in X21.

P Input parameter.

P is INTEGER

The number of rows in X11. 0 <= P <= M.

Q Input parameter.

Q is INTEGER

The number of columns in X11 and X21. 0 <= Q <= MIN(P,M-P,M-Q).

X11 Input and output parameter.

X11 is COMPLEX*16

X11 is an array, dimension (LDX11, Q). On entry, the top block of the matrix X to be reduced. On exit, the columns of tril(X11) specify reflectors for P1 and the rows of triu(X11,1) specify reflectors for Q1.

LDX11 Input parameter.

LDX11 is INTEGER

The leading dimension of X11. LDX11 >= P.

X21 Input and output parameter.

X21 is COMPLEX*16

X21 is an array, dimension (LDX21, Q). On entry, the bottom block of the matrix X to be reduced. On exit, the columns of tril(X21) specify reflectors for P2.

LDX21 Input parameter.

LDX21 is INTEGER

The leading dimension of X21. LDX21 >= M-P.

THETA Output parameter.

THETA is DOUBLE PRECISION

THETA is an array, dimension (Q). The entries of the bidiagonal blocks B11, B21 are defined by THETA and PHI. See Further Details.

PHI Output parameter.

PHI is DOUBLE PRECISION

PHI is an array, dimension (Q-1). The entries of the bidiagonal blocks B11, B21 are defined by THETA and PHI. See Further Details.

TAUP1 Output parameter.

TAUP1 is COMPLEX*16

TAUP1 is an array, dimension (P). The scalar factors of the elementary reflectors that define P1.

TAUP2 Output parameter.

TAUP2 is COMPLEX*16

TAUP2 is an array, dimension (M-P). The scalar factors of the elementary reflectors that define P2.

TAUQ1 Output parameter.

TAUQ1 is COMPLEX*16

TAUQ1 is an array, dimension (Q). The scalar factors of the elementary reflectors that define Q1.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= M-Q.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cunbdb1*.

4.17.801 zunbdb2

zunbdb2 simultaneously bidiagonalizes the blocks of a tall and skinny matrix X with orthonomal columns:

```
                         [ B11 ]
[ X11 ]    [ P1 |    ] [  0  ]
[-----] = [----------] [-----] Q1**T .
[ X21 ]    [    | P2 ] [ B21 ]
                         [  0  ]
```

X11 is P-by-Q, and X21 is (M-P)-by-Q. P must be no larger than M-P, Q, or M-Q. Routines ZUNBDB1, ZUNBDB3, and ZUNBDB4 handle cases in which P is not the minimum dimension.

The unitary matrices P1, P2, and Q1 are P-by-P, (M-P)-by-(M-P), and (M-Q)-by-(M-Q), respectively. They are represented implicitly by Householder vectors.

B11 and B12 are P-by-P bidiagonal matrices represented implicitly by angles THETA, PHI.

Syntax

Fortran specification:

```
use armpl_library

subroutine zunbdb2(M, P, Q, X11, LDX11, X21, LDX21, THETA, PHI, TAUP1, TAUP2,
                   TAUQ1, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zunbdb2_(const armpl_int_t *m, const armpl_int_t *p,
              const armpl_int_t *q, armpl_doublecomplex_t *x11,
              const armpl_int_t *ldx11, armpl_doublecomplex_t *x21,
              const armpl_int_t *ldx21, double *theta, double *phi,
              armpl_doublecomplex_t *taup1, armpl_doublecomplex_t *taup2,
              armpl_doublecomplex_t *tauq1, armpl_doublecomplex_t *work,
              const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows X11 plus the number of rows in X21.

P Input parameter.

P is INTEGER

The number of rows in X11. 0 <= P <= min(M-P, Q,M-Q).

Q Input parameter.

Q is INTEGER

The number of columns in X11 and X21. 0 <= Q <= M.

X11 Input and output parameter.

X11 is COMPLEX*16

X11 is an array, dimension (LDX11, Q). On entry, the top block of the matrix X to be reduced. On exit, the columns of tril(X11) specify reflectors for P1 and the rows of triu(X11,1) specify reflectors for Q1.

LDX11 Input parameter.

LDX11 is INTEGER

The leading dimension of X11. LDX11 >= P.

X21 Input and output parameter.

X21 is COMPLEX*16

X21 is an array, dimension (LDX21, Q). On entry, the bottom block of the matrix X to be reduced. On exit, the columns of tril(X21) specify reflectors for P2.

LDX21 Input parameter.

LDX21 is INTEGER

The leading dimension of X21. LDX21 >= M-P.

THETA Output parameter.

THETA is DOUBLE PRECISION

THETA is an array, dimension (Q). The entries of the bidiagonal blocks B11, B21 are defined by THETA and PHI. See Further Details.

PHI Output parameter.

PHI is DOUBLE PRECISION

PHI is an array, dimension (Q-1). The entries of the bidiagonal blocks B11, B21 are defined by THETA and PHI. See Further Details.

TAUP1 Output parameter.

TAUP1 is COMPLEX*16

TAUP1 is an array, dimension (P). The scalar factors of the elementary reflectors that define P1.

TAUP2 Output parameter.

TAUP2 is COMPLEX*16

TAUP2 is an array, dimension (M-P). The scalar factors of the elementary reflectors that define P2.

TAUQ1 Output parameter.

TAUQ1 is COMPLEX*16

TAUQ1 is an array, dimension (Q). The scalar factors of the elementary reflectors that define Q1.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= M-Q.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cunbdb2*.

4.17.802 zunbdb3

zunbdb3 simultaneously bidiagonalizes the blocks of a tall and skinny matrix X with orthonomal columns:

```
                         [ B11 ]
[ X11 ]   [ P1 |    ] [  0  ]
[-----] = [---------] [-----] Q1**T .
[ X21 ]   [    | P2 ] [ B21 ]
                         [  0  ]
```

X11 is P-by-Q, and X21 is (M-P)-by-Q. M-P must be no larger than P, Q, or M-Q. Routines ZUNBDB1, ZUNBDB2, and ZUNBDB4 handle cases in which M-P is not the minimum dimension.

The unitary matrices P1, P2, and Q1 are P-by-P, (M-P)-by-(M-P), and (M-Q)-by-(M-Q), respectively. They are represented implicitly by Householder vectors.

B11 and B12 are (M-P)-by-(M-P) bidiagonal matrices represented implicitly by angles THETA, PHI.

Syntax

Fortran specification:

```
use armpl_library

subroutine zunbdb3(M, P, Q, X11, LDX11, X21, LDX21, THETA, PHI, TAUP1, TAUP2,
                   TAUQ1, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zunbdb3_(const armpl_int_t *m, const armpl_int_t *p,
              const armpl_int_t *q, armpl_doublecomplex_t *x11,
              const armpl_int_t *ldx11, armpl_doublecomplex_t *x21,
              const armpl_int_t *ldx21, double *theta, double *phi,
              armpl_doublecomplex_t *taup1, armpl_doublecomplex_t *taup2,
              armpl_doublecomplex_t *tauq1, armpl_doublecomplex_t *work,
              const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows X11 plus the number of rows in X21.

P Input parameter.

P is INTEGER

The number of rows in X11. 0 <= P <= M. M-P <= min(P, Q,M-Q).

Q Input parameter.

Q is INTEGER

The number of columns in X11 and X21. 0 <= Q <= M.

X11 Input and output parameter.

X11 is COMPLEX*16

X11 is an array, dimension (LDX11, Q). On entry, the top block of the matrix X to be reduced. On exit, the columns of tril(X11) specify reflectors for P1 and the rows of triu(X11,1) specify reflectors for Q1.

LDX11 Input parameter.

LDX11 is INTEGER

The leading dimension of X11. LDX11 >= P.

X21 Input and output parameter.

X21 is COMPLEX*16

X21 is an array, dimension (LDX21, Q). On entry, the bottom block of the matrix X to be reduced. On exit, the columns of tril(X21) specify reflectors for P2.

LDX21 Input parameter.

LDX21 is INTEGER

The leading dimension of X21. LDX21 >= M-P.

THETA Output parameter.

THETA is DOUBLE PRECISION

THETA is an array, dimension (Q). The entries of the bidiagonal blocks B11, B21 are defined by THETA and PHI. See Further Details.

PHI Output parameter.

PHI is DOUBLE PRECISION

PHI is an array, dimension (Q-1). The entries of the bidiagonal blocks B11, B21 are defined by THETA and PHI. See Further Details.

TAUP1 Output parameter.

TAUP1 is COMPLEX*16

TAUP1 is an array, dimension (P). The scalar factors of the elementary reflectors that define P1.

TAUP2 Output parameter.

TAUP2 is COMPLEX*16

TAUP2 is an array, dimension (M-P). The scalar factors of the elementary reflectors that define P2.

TAUQ1 Output parameter.

TAUQ1 is COMPLEX*16

TAUQ1 is an array, dimension (Q). The scalar factors of the elementary reflectors that define Q1.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= M-Q.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cunbdb3*.

4.17.803 zunbdb4

zunbdb4 simultaneously bidiagonalizes the blocks of a tall and skinny matrix X with orthonomal columns:

```
                         [ B11 ]
[ X11 ]   [ P1 |    ] [  0  ]
[-----] = [---------] [-----] Q1**T .
[ X21 ]   [    | P2 ] [ B21 ]
                         [  0  ]
```

X11 is P-by-Q, and X21 is (M-P)-by-Q. M-Q must be no larger than P, M-P, or Q. Routines ZUNBDB1, ZUNBDB2, and ZUNBDB3 handle cases in which M-Q is not the minimum dimension.

The unitary matrices P1, P2, and Q1 are P-by-P, (M-P)-by-(M-P), and (M-Q)-by-(M-Q), respectively. They are represented implicitly by Householder vectors.

B11 and B12 are (M-Q)-by-(M-Q) bidiagonal matrices represented implicitly by angles THETA, PHI.

Syntax

Fortran specification:

```
use armpl_library

subroutine zunbdb4(M, P, Q, X11, LDX11, X21, LDX21, THETA, PHI, TAUP1, TAUP2,
                   TAUQ1, PHANTOM, WORK, LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zunbdb4_(const armpl_int_t *m, const armpl_int_t *p,
              const armpl_int_t *q, armpl_doublecomplex_t *x11,
              const armpl_int_t *ldx11, armpl_doublecomplex_t *x21,
              const armpl_int_t *ldx21, double *theta, double *phi,
              armpl_doublecomplex_t *taup1, armpl_doublecomplex_t *taup2,
              armpl_doublecomplex_t *tauq1, armpl_doublecomplex_t *phantom,
              armpl_doublecomplex_t *work, const armpl_int_t *lwork,
              armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows X11 plus the number of rows in X21.

P Input parameter.

P is INTEGER

The number of rows in X11. 0 <= P <= M.

Q Input parameter.

Q is INTEGER

The number of columns in X11 and X21. 0 <= Q <= M and M-Q <= min(P,M-P, Q).

X11 Input and output parameter.

X11 is COMPLEX*16

X11 is an array, dimension (LDX11, Q). On entry, the top block of the matrix X to be reduced. On exit, the columns of tril(X11) specify reflectors for P1 and the rows of triu(X11,1) specify reflectors for Q1.

LDX11 Input parameter.

LDX11 is INTEGER

The leading dimension of X11. LDX11 >= P.

X21 Input and output parameter.

X21 is COMPLEX*16

X21 is an array, dimension (LDX21, Q). On entry, the bottom block of the matrix X to be reduced. On exit, the columns of tril(X21) specify reflectors for P2.

LDX21 Input parameter.

LDX21 is INTEGER

The leading dimension of X21. LDX21 >= M-P.

THETA Output parameter.

THETA is DOUBLE PRECISION

THETA is an array, dimension (Q). The entries of the bidiagonal blocks B11, B21 are defined by THETA and PHI. See Further Details.

PHI Output parameter.

PHI is DOUBLE PRECISION

PHI is an array, dimension (Q-1). The entries of the bidiagonal blocks B11, B21 are defined by THETA and PHI. See Further Details.

TAUP1 Output parameter.

TAUP1 is COMPLEX*16

TAUP1 is an array, dimension (P). The scalar factors of the elementary reflectors that define P1.

TAUP2 Output parameter.

TAUP2 is COMPLEX*16

TAUP2 is an array, dimension (M-P). The scalar factors of the elementary reflectors that define P2.

TAUQ1 Output parameter.

TAUQ1 is COMPLEX*16

TAUQ1 is an array, dimension (Q). The scalar factors of the elementary reflectors that define Q1.

PHANTOM Output parameter.

PHANTOM is COMPLEX*16

PHANTOM is an array, dimension (M). The routine computes an M-by-1 column vector Y that is orthogonal to the columns of [X11; X21]. PHANTOM(1:P) and PHANTOM(P+1:M) contain Householder vectors for Y(1:P) and Y(P+1:M), respectively.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= M-Q.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cunbdb4*.

4.17.804 zunbdb5

zunbdb5 orthogonalizes the column vector

```
X = [ X1 ]
    [ X2 ]
```

with respect to the columns of

```
Q = [ Q1 ] .
    [ Q2 ]
```

The columns of Q must be orthonormal.

If the projection is zero according to Kahan's "twice is enough" criterion, then some other vector from the orthogonal complement is returned. This vector is chosen in an arbitrary but deterministic way.

Syntax

Fortran specification:

```
use armpl_library

subroutine zunbdb5(M1, M2, N, X1, INCX1, X2, INCX2, Q1, LDQ1, Q2, LDQ2, WORK,
                   LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zunbdb5_(const armpl_int_t *m1, const armpl_int_t *m2,
              const armpl_int_t *n, armpl_doublecomplex_t *x1,
              const armpl_int_t *incx1, armpl_doublecomplex_t *x2,
              const armpl_int_t *incx2, armpl_doublecomplex_t *q1,
              const armpl_int_t *ldq1, armpl_doublecomplex_t *q2,
              const armpl_int_t *ldq2, armpl_doublecomplex_t *work,
              const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M1 Input parameter.

M1 is INTEGER

The dimension of X1 and the number of rows in Q1. 0 <= M1.

M2 Input parameter.

M2 is INTEGER

The dimension of X2 and the number of rows in Q2. 0 <= M2.

N Input parameter.

N is INTEGER

The number of columns in Q1 and Q2. 0 <= N.

X1 Input and output parameter.

X1 is COMPLEX*16

X1 is an array, dimension (M1). On entry, the top part of the vector to be orthogonalized. On exit, the top part of the projected vector.

INCX1 Input parameter.

INCX1 is INTEGER

Increment for entries of X1.

X2 Input and output parameter.

X2 is COMPLEX*16

X2 is an array, dimension (M2). On entry, the bottom part of the vector to be orthogonalized. On exit, the bottom part of the projected vector.

INCX2 Input parameter.

INCX2 is INTEGER

Increment for entries of X2.

Q1 Input parameter.

Q1 is COMPLEX*16

Q1 is an array, dimension (LDQ1, N). The top part of the orthonormal basis matrix.

LDQ1 Input parameter.

LDQ1 is INTEGER

The leading dimension of Q1. LDQ1 >= M1.

Q2 Input parameter.

Q2 is COMPLEX*16

Q2 is an array, dimension (LDQ2, N). The bottom part of the orthonormal basis matrix.

LDQ2 Input parameter.

LDQ2 is INTEGER

The leading dimension of Q2. LDQ2 >= M2.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= N.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cunbdb5*.

4.17.805 zunbdb6

zunbdb6 orthogonalizes the column vector

```
X = [ X1 ]
    [ X2 ]
```

with respect to the columns of

```
Q = [ Q1 ] .
    [ Q2 ]
```

The columns of Q must be orthonormal.

If the projection is zero according to Kahan's "twice is enough" criterion, then the zero vector is returned.

Syntax

Fortran specification:

```
use armpl_library

subroutine zunbdb6(M1, M2, N, X1, INCX1, X2, INCX2, Q1, LDQ1, Q2, LDQ2, WORK,
                   LWORK, INFO)
```

C specification:

```
#include "armpl.h"

void zunbdb6_(const armpl_int_t *m1, const armpl_int_t *m2,
              const armpl_int_t *n, armpl_doublecomplex_t *x1,
              const armpl_int_t *incx1, armpl_doublecomplex_t *x2,
              const armpl_int_t *incx2, armpl_doublecomplex_t *q1,
              const armpl_int_t *ldq1, armpl_doublecomplex_t *q2,
              const armpl_int_t *ldq2, armpl_doublecomplex_t *work,
              const armpl_int_t *lwork, armpl_int_t *info);
```

Parameters

M1 Input parameter.

M1 is INTEGER

The dimension of X1 and the number of rows in Q1. 0 <= M1.

M2 Input parameter.

M2 is INTEGER

The dimension of X2 and the number of rows in Q2. 0 <= M2.

N Input parameter.

N is INTEGER

The number of columns in Q1 and Q2. 0 <= N.

X1 Input and output parameter.

X1 is COMPLEX*16

X1 is an array, dimension (M1). On entry, the top part of the vector to be orthogonalized. On exit, the top part of the projected vector.

INCX1 Input parameter.

INCX1 is INTEGER

Increment for entries of X1.

X2 Input and output parameter.

X2 is COMPLEX*16

X2 is an array, dimension (M2). On entry, the bottom part of the vector to be orthogonalized. On exit, the bottom part of the projected vector.

INCX2 Input parameter.

INCX2 is INTEGER

Increment for entries of X2.

Q1 Input parameter.

Q1 is COMPLEX*16

Q1 is an array, dimension (LDQ1, N). The top part of the orthonormal basis matrix.

LDQ1 Input parameter.

LDQ1 is INTEGER

The leading dimension of Q1. LDQ1 >= M1.

Q2 Input parameter.

Q2 is COMPLEX*16

Q2 is an array, dimension (LDQ2, N). The bottom part of the orthonormal basis matrix.

LDQ2 Input parameter.

LDQ2 is INTEGER

The leading dimension of Q2. LDQ2 >= M2.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (LWORK) .

LWORK Input parameter.

LWORK is INTEGER

The dimension of the array WORK. LWORK >= N.

INFO Output parameter.

INFO is INTEGER

= 0: successful exit. < 0: if INFO = -i, the i-th argument had an illegal value.

Related Information

For this routine in other precisions, please see *cunbdb6*.

4.17.806 zung2l

zung2l generates an m by n complex matrix Q with orthonormal columns, which is defined as the last n columns of a product of k elementary reflectors of order m

```
Q  =  H(k) . . . H(2) H(1)
```

as returned by ZGEQLF.

Syntax

Fortran specification:

```
use armpl_library

subroutine zung2l(M, N, K, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zung2l_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *k,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_doublecomplex_t *tau, armpl_doublecomplex_t *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix Q. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix Q. M >= N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. N >= K >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the (n-k+i)-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by ZGEQLF in the last k columns of its array argument A. On exit, the m-by-n matrix Q.

LDA Input parameter.

LDA is INTEGER

The first dimension of the array A. LDA >= max(1, M).

TAU Input parameter.

TAU is COMPLEX*16

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by ZGEQLF.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument has an illegal value

Related Information

For this routine in other precisions, please see *cung2l*.

4.17.807 zung2r

zung2r generates an m by n complex matrix Q with orthonormal columns, which is defined as the first n columns of a product of k elementary reflectors of order m

```
Q  =  H(1) H(2) . . . H(k)
```

as returned by ZGEQRF.

Syntax

Fortran specification:

```
use armpl_library

subroutine zung2r(M, N, K, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zung2r_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *k,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_doublecomplex_t *tau, armpl_doublecomplex_t *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix Q. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix Q. M >= N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. N >= K >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the i-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,. . . ,k, as returned by ZGEQRF in the first k columns of its array argument A. On exit, the m by n matrix Q.

LDA Input parameter.

LDA is INTEGER

The first dimension of the array A. LDA >= max(1, M).

TAU Input parameter.

TAU is COMPLEX*16

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by ZGEQRF.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (N) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument has an illegal value

Related Information

For this routine in other precisions, please see *cung2r*.

4.17.808 zungl2

`zungl2` generates an m-by-n complex matrix Q with orthonormal rows, which is defined as the first m rows of a product of k elementary reflectors of order n

```
Q  =  H(k)**H . . . H(2)**H H(1)**H
```

as returned by ZGELQF.

Syntax

Fortran specification:

```
use armpl_library

subroutine zungl2(M, N, K, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zungl2_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *k,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_doublecomplex_t *tau, armpl_doublecomplex_t *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

`M` is `INTEGER`

The number of rows of the matrix Q. `M` >= 0.

N Input parameter.

`N` is `INTEGER`

The number of columns of the matrix Q. `N` >= `M`.

K Input parameter.

`K` is `INTEGER`

The number of elementary reflectors whose product defines the matrix Q. `M` >= `K` >= 0.

A Input and output parameter.

`A` is `COMPLEX*16`

`A` is an array, dimension (`LDA`, `N`). On entry, the i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by ZGELQF in the first k rows of its array argument `A`. On exit, the m by n matrix Q.

LDA Input parameter.

`LDA` is `INTEGER`

The first dimension of the array `A`. `LDA` >= max(1, `M`).

TAU Input parameter.

TAU is COMPLEX*16

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by ZGELQF.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (M) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument has an illegal value

Related Information

For this routine in other precisions, please see *cungl2*.

4.17.809 zungr2

zungr2 generates an m by n complex matrix Q with orthonormal rows, which is defined as the last m rows of a product of k elementary reflectors of order n

```
Q  =  H(1)**H H(2)**H . . . H(k)**H
```

as returned by ZGERQF.

Syntax

Fortran specification:

```
use armpl_library

subroutine zungr2(M, N, K, A, LDA, TAU, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zungr2_(const armpl_int_t *m, const armpl_int_t *n, const armpl_int_t *k,
             armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_doublecomplex_t *tau, armpl_doublecomplex_t *work,
             armpl_int_t *info);
```

Parameters

M Input parameter.

M is INTEGER

The number of rows of the matrix Q. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix Q. N >= M.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. M >= K >= 0.

A Input and output parameter.

A is COMPLEX*16

A is an array, dimension (LDA, N). On entry, the (m-k+i)-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by ZGERQF in the last k rows of its array argument A. On exit, the m-by-n matrix Q.

LDA Input parameter.

LDA is INTEGER

The first dimension of the array A. LDA >= max(1, M).

TAU Input parameter.

TAU is COMPLEX*16

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by ZGERQF.

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension (M) .

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument has an illegal value

Related Information

For this routine in other precisions, please see *cungr2*.

4.17.810 zunm2l

zunm2l overwrites the general complex m-by-n matrix C with

```
Q * C  if SIDE = 'L' and TRANS = 'N', or

Q**H* C  if SIDE = 'L' and TRANS = 'C', or

C * Q  if SIDE = 'R' and TRANS = 'N', or

C * Q**H if SIDE = 'R' and TRANS = 'C',
```

where Q is a complex unitary matrix defined as the product of k elementary reflectors

```
Q = H(k) . . . H(2) H(1)
```

as returned by ZGEQLF. Q is of order m if SIDE = 'L' and of order n if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine zunm2l(SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zunm2l_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_doublecomplex_t *tau, armpl_doublecomplex_t *c,
             const armpl_int_t *ldc, armpl_doublecomplex_t *work,
             armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left = 'R': apply Q or Q^H from the Right

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply Q (No transpose) = 'C': apply Q^H (Conjugate transpose)

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, K). The i-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by ZGEQLF in the last k columns of its array argument A. A is modified by the routine but restored on exit.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDA >= max(1, M); if SIDE = 'R', LDA >= max(1, N).

TAU Input parameter.

TAU is COMPLEX*16

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by ZGEQLF.

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). On entry, the m-by-n matrix C. On exit, C is overwritten by Q* C or Q^H * C or C*Q^H or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension. (N) if SIDE = 'L', (M) if SIDE = 'R'

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cunm2l*.

4.17.811 zunm2r

zunm2r overwrites the general complex m-by-n matrix C with

```
Q * C  if SIDE = 'L' and TRANS = 'N', or

Q**H* C  if SIDE = 'L' and TRANS = 'C', or

C * Q  if SIDE = 'R' and TRANS = 'N', or

C * Q**H if SIDE = 'R' and TRANS = 'C',
```

where Q is a complex unitary matrix defined as the product of k elementary reflectors

```
Q = H(1) H(2) . . . H(k)
```

as returned by ZGEQRF. Q is of order m if SIDE = 'L' and of order n if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine zunm2r(SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zunm2r_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_doublecomplex_t *tau, armpl_doublecomplex_t *c,
             const armpl_int_t *ldc, armpl_doublecomplex_t *work,
             armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left = 'R': apply Q or Q^H from the Right

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply Q (No transpose) = 'C': apply Q^H (Conjugate transpose)

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension (LDA, K). The i-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by ZGEQRF in the first k columns of its array argument A. A is modified by the routine but restored on exit.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. If SIDE = 'L', LDA >= max(1, M); if SIDE = 'R', LDA >= max(1, N).

TAU Input parameter.

TAU is COMPLEX*16

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by ZGEQRF.

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). On entry, the m-by-n matrix C. On exit, C is overwritten by Q* C or Q^H * C or C*Q^H or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension. (N) if SIDE = 'L', (M) if SIDE = 'R'

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cunm2r*.

4.17.812 zunml2

zunml2 overwrites the general complex m-by-n matrix C with

```
Q * C  if SIDE = 'L' and TRANS = 'N', or

Q**H* C  if SIDE = 'L' and TRANS = 'C', or

C * Q  if SIDE = 'R' and TRANS = 'N', or

C * Q**H if SIDE = 'R' and TRANS = 'C',
```

where Q is a complex unitary matrix defined as the product of k elementary reflectors

```
Q = H(k)**H . . . H(2)**H H(1)**H
```

as returned by ZGELQF. Q is of order m if SIDE = 'L' and of order n if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine zunml2(SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zunml2_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_doublecomplex_t *tau, armpl_doublecomplex_t *c,
             const armpl_int_t *ldc, armpl_doublecomplex_t *work,
             armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left = 'R': apply Q or Q^H from the Right

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply Q (No transpose) = 'C': apply Q^H (Conjugate transpose)

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension. (LDA, M) if SIDE = 'L', (LDA, N) if SIDE = 'R' The i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by ZGELQF in the first k rows of its array argument A. A is modified by the routine but restored on exit.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, K).

TAU Input parameter.

TAU is COMPLEX*16

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by ZGELQF.

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). On entry, the m-by-n matrix C. On exit, C is overwritten by Q* C or Q^H * C or C*Q^H or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension. (N) if SIDE = 'L', (M) if SIDE = 'R'

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cunml2*.

4.17.813 zunmr2

zunmr2 overwrites the general complex m-by-n matrix C with

```
Q * C  if SIDE = 'L' and TRANS = 'N', or

Q**H* C  if SIDE = 'L' and TRANS = 'C', or

C * Q  if SIDE = 'R' and TRANS = 'N', or

C * Q**H if SIDE = 'R' and TRANS = 'C',
```

where Q is a complex unitary matrix defined as the product of k elementary reflectors

```
Q = H(1)**H H(2)**H . . . H(k)**H
```

as returned by ZGERQF. Q is of order m if SIDE = 'L' and of order n if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine zunmr2(SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zunmr2_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_doublecomplex_t *tau, armpl_doublecomplex_t *c,
             const armpl_int_t *ldc, armpl_doublecomplex_t *work,
             armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left = 'R': apply Q or Q^H from the Right

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply Q (No transpose) = 'C': apply Q^H (Conjugate transpose)

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension. (LDA, M) if SIDE = 'L', (LDA, N) if SIDE = 'R' The i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by ZGERQF in the last k rows of its array argument A. A is modified by the routine but restored on exit.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, K).

TAU Input parameter.

TAU is COMPLEX*16

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by ZGERQF.

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). On entry, the m-by-n matrix C. On exit, C is overwritten by Q* C or Q^H * C or C*Q^H or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension. (N) if SIDE = 'L', (M) if SIDE = 'R'

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cunmr2*.

4.17.814 zunmr3

zunmr3 overwrites the general complex m by n matrix C with

```
Q * C  if SIDE = 'L' and TRANS = 'N', or

Q**H* C  if SIDE = 'L' and TRANS = 'C', or

C * Q  if SIDE = 'R' and TRANS = 'N', or

C * Q**H if SIDE = 'R' and TRANS = 'C',
```

where Q is a complex unitary matrix defined as the product of k elementary reflectors

```
Q = H(1) H(2) . . . H(k)
```

as returned by ZTZRZF. Q is of order m if SIDE = 'L' and of order n if SIDE = 'R'.

Syntax

Fortran specification:

```
use armpl_library

subroutine zunmr3(SIDE, TRANS, M, N, K, L, A, LDA, TAU, C, LDC, WORK, INFO)
```

C specification:

```
#include "armpl.h"

void zunmr3_(const char *side, const char *trans, const armpl_int_t *m,
             const armpl_int_t *n, const armpl_int_t *k, const armpl_int_t *l,
             const armpl_doublecomplex_t *a, const armpl_int_t *lda,
             const armpl_doublecomplex_t *tau, armpl_doublecomplex_t *c,
             const armpl_int_t *ldc, armpl_doublecomplex_t *work,
             armpl_int_t *info, ... );
```

Parameters

SIDE Input parameter.

SIDE is CHARACTER*1

= 'L': apply Q or Q^H from the Left = 'R': apply Q or Q^H from the Right

TRANS Input parameter.

TRANS is CHARACTER*1

= 'N': apply Q (No transpose) = 'C': apply Q^H (Conjugate transpose)

M Input parameter.

M is INTEGER

The number of rows of the matrix C. M >= 0.

N Input parameter.

N is INTEGER

The number of columns of the matrix C. N >= 0.

K Input parameter.

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. If SIDE = 'L', M >= K >= 0; if SIDE = 'R', N >= K >= 0.

L Input parameter.

L is INTEGER

The number of columns of the matrix A containing the meaningful part of the Householder reflectors. If SIDE = 'L', M >= L >= 0, if SIDE = 'R', N >= L >= 0.

A Input parameter.

A is COMPLEX*16

A is an array, dimension. (LDA, M) if SIDE = 'L', (LDA, N) if SIDE = 'R' The i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by ZTZRZF in the last k rows of its array argument A. A is modified by the routine but restored on exit.

LDA Input parameter.

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1, K).

TAU Input parameter.

TAU is COMPLEX*16

TAU is an array, dimension (K). TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by ZTZRZF.

C Input and output parameter.

C is COMPLEX*16

C is an array, dimension (LDC, N). On entry, the m-by-n matrix C. On exit, C is overwritten by Q* C or Q^H * C or C*Q^H or C*Q.

LDC Input parameter.

LDC is INTEGER

The leading dimension of the array C. LDC >= max(1, M).

WORK Output parameter.

WORK is COMPLEX*16

WORK is an array, dimension. (N) if SIDE = 'L', (M) if SIDE = 'R'

INFO Output parameter.

INFO is INTEGER

= 0: successful exit < 0: if INFO = -i, the i-th argument had an illegal value

Related Information

For this routine in other precisions, please see *cunmr3*.

4.18 LAPACK utilities routines

4.18.1 chla_transtype

This subroutine translates from a BLAST-specified integer constant to the character string specifying a transposition operation.

CHLA_TRANSTYPE returns an CHARACTER*1. If CHLA_TRANSTYPE is 'X', then input is not an integer indicating a transposition operator. Otherwise CHLA_TRANSTYPE returns the constant value corresponding to TRANS.

Syntax

Fortran specification:

```
use armpl_library

character*1 function chla_transtype(TRANS)
```

C specification:

```
#include "armpl.h"

char chla_transtype_(const armpl_int_t *trans);
```

Related Information

4.18.2 dlabad

`dlabad` takes as input the values computed by DLAMCH for underflow and overflow, and returns the square root of each of these values if the log of LARGE is sufficiently large. This subroutine is intended to identify machines with a large exponent range, such as the Crays, and redefine the underflow and overflow limits to be the square roots of the values computed by DLAMCH. This subroutine is needed because DLAMCH does not compensate for poor arithmetic in the upper half of the exponent range, as is found on a Cray.

Syntax

Fortran specification:

```
use armpl_library

subroutine dlabad(SMALL, LARGE)
```

C specification:

```
#include "armpl.h"

void dlabad_(double *small, double *large);
```

Parameters

SMALL Input and output parameter.

`SMALL` is `DOUBLE PRECISION`

On entry, the underflow threshold as computed by DLAMCH. On exit, if LOG10(`LARGE`) is sufficiently large, the square root of `SMALL`, otherwise unchanged.

LARGE Input and output parameter.

LARGE is DOUBLE PRECISION

On entry, the overflow threshold as computed by DLAMCH. On exit, if LOG10(LARGE) is sufficiently large, the square root of LARGE, otherwise unchanged.

Related Information

For this routine in other precisions, please see *slabad*.

4.18.3 ieeeck

ieeeck is called from the ILAENV to verify that Infinity and possibly NaN arithmetic is safe (i.e. will not trap).

Syntax

Fortran specification:

```
use armpl_library

integer function ieeeck(ISPEC, ZERO, ONE)
```

C specification:

```
#include "armpl.h"

armpl_int_t ieeeck_(const armpl_int_t *ispec, const float *zero,
                    const float *one);
```

Parameters

ISPEC Input parameter.

ISPEC is INTEGER

Specifies whether to test just for inifinity arithmetic or whether to test for infinity and NaN arithmetic. = 0: Verify infinity arithmetic only. = 1: Verify infinity and NaN arithmetic.

ZERO Input parameter.

ZERO is REAL

Must contain the value 0.0 This is passed to prevent the compiler from optimizing away this code.

ONE Input parameter.

ONE is REAL

Must contain the value 1.0 This is passed to prevent the compiler from optimizing away this code.

RETURN VALUE: INTEGER = 0: Arithmetic failed to produce the correct answers = 1: Arithmetic produced the correct answers

Related Information

4.18.4 iladiag

This subroutine translated from a character string specifying if a matrix has unit diagonal or not to the relevant BLAST-specified integer constant.

iladiag returns an INTEGER. If iladiag < 0, then the input is not a character indicating a unit or non-unit diagonal. Otherwise iladiag returns the constant value corresponding to DIAG.

Syntax

Fortran specification:

```
use armpl_library

integer function iladiag(DIAG)
```

C specification:

```
#include "armpl.h"

armpl_int_t iladiag_(const char *diag, ... );
```

Related Information

4.18.5 ilaenv

ilaenv is called from the LAPACK routines to choose problem-dependent parameters for the local environment. See ISPEC for a description of the parameters.

ilaenv returns an INTEGER if ilaenv >= 0: ilaenv returns the value of the parameter specified by ISPEC if ilaenv < 0: if ilaenv = -k, the k-th argument had an illegal value.

This version provides a set of parameters which should give good, but not optimal, performance on many of the currently available computers. Users are encouraged to modify this subroutine to set the tuning parameters for their particular machine using the option and problem size information in the arguments.

This routine will not function correctly if it is converted to all lower case. Converting it to all upper case is allowed.

Syntax

Fortran specification:

```
use armpl_library

integer function ilaenv(ISPEC, NAME, OPTS, N1, N2, N3, N4)
```

C specification:

```
#include "armpl.h"

armpl_int_t ilaenv_(const armpl_int_t *ispec, const char *name,
                    const char *opts, const armpl_int_t *n1,
                    const armpl_int_t *n2, const armpl_int_t *n3,
                    const armpl_int_t *n4, ... );
```

Parameters

ISPEC Input parameter.

ISPEC is INTEGER

Specifies the parameter to be returned as the value of ILAENV. = 1: the optimal blocksize; if this value is 1, an unblocked algorithm will give the best performance. = 2: the minimum block size for which the block routine should be used; if the usable block size is less than this value, an unblocked routine should be used. =

3: the crossover point (in a block routine, for N less than this value, an unblocked routine should be used) = 4: the number of shifts, used in the nonsymmetric eigenvalue routines (DEPRECATED) = 5: the minimum column dimension for blocking to be used; rectangular blocks must have dimension at least k by m, where k is given by ILAENV(2,...) and m by ILAENV(5,...) = 6: the crossover point for the SVD (when reducing an m by n matrix to bidiagonal form, if max(m,n)/min(m,n) exceeds this value, a QR factorization is used first to reduce the matrix to a triangular form.) = 7: the number of processors = 8: the crossover point for the multishift QR method for nonsymmetric eigenvalue problems (DEPRECATED) = 9: maximum size of the subproblems at the bottom of the computation tree in the divide-and-conquer algorithm (used by xGELSD and xGESDD) =10: ieee NaN arithmetic can be trusted not to trap =11: infinity arithmetic can be trusted not to trap 12 <= `ISPEC` <= 16: xHSEQR or related subroutines, see IPARMQ for detailed explanation

NAME Input parameter.

`NAME` is CHARACTER*(*)

The name of the calling subroutine, in either upper case or lower case.

OPTS Input parameter.

`OPTS` is CHARACTER*(*)

The character options to the subroutine `NAME`, concatenated into a single character string. For example, UPLO = 'U', TRANS = 'T', and DIAG = 'N' for a triangular routine would be specified as `OPTS` = 'UTN'.

N1 Input parameter.

`N1` is `INTEGER`

N2 Input parameter.

`N2` is `INTEGER`

N3 Input parameter.

`N3` is `INTEGER`

N4 Input parameter.

`N4` is `INTEGER`

Problem dimensions for the subroutine NAME; these may not all be required.

Related Information

4.18.6 ilaprec

This subroutine translated from a character string specifying an intermediate precision to the relevant BLAST-specified integer constant.

`ilaprec` returns an INTEGER. If `ilaprec` < 0, then the input is not a character indicating a supported intermediate precision. Otherwise `ilaprec` returns the constant value corresponding to PREC.

Syntax

Fortran specification:

```
use armpl_library

integer function ilaprec(PREC)
```

C specification:

```
#include "armpl.h"

armpl_int_t ilaprec_(const char *prec, ... );
```

Related Information

4.18.7 ilatrans

This subroutine translates from a character string specifying a transposition operation to the relevant BLAST-specified integer constant.

ILATRANS returns an INTEGER. If ILATRANS < 0, then the input is not a character indicating a transposition operator. Otherwise ILATRANS returns the constant value corresponding to TRANS.

Syntax

Fortran specification:

```
use armpl_library

integer function ilatrans(TRANS)
```

C specification:

```
#include "armpl.h"

armpl_int_t ilatrans_(const char *trans, ... );
```

Related Information

4.18.8 ilauplo

This subroutine translated from a character string specifying a upper- or lower-triangular matrix to the relevant BLAST-specified integer constant.

`ilauplo` returns an INTEGER. If `ilauplo` < 0, then the input is not a character indicating an upper- or lower-triangular matrix. Otherwise `ilauplo` returns the constant value corresponding to UPLO.

Syntax

Fortran specification:

```
use armpl_library

integer function ilauplo(UPLO)
```

C specification:

```
#include "armpl.h"

armpl_int_t ilauplo_(const char *uplo, ... );
```

Related Information

4.18.9 iparmq

```
This program sets problem and machine dependent parameters
     useful for xHSEQR and related subroutines for eigenvalue
     problems. It is called whenever
     IPARMQ is called with 12 <= ISPEC <= 16
```

Syntax

Fortran specification:

```
use armpl_library

integer function iparmq(ISPEC, NAME, OPTS, N, ILO, IHI, LWORK)
```

C specification:

```
#include "armpl.h"

armpl_int_t iparmq_(const armpl_int_t *ispec, const char *name,
                    const char *opts, const armpl_int_t *n,
                    const armpl_int_t *ilo, const armpl_int_t *ihi,
                    const armpl_int_t *lwork, ... );
```

Parameters

ISPEC Input parameter.

ISPEC is INTEGER

ISPEC specifies which tunable parameter IPARMQ should return.

ISPEC=12: (INMIN) Matrices of order nmin or less are sent directly to xLAHQR, the implicit double shift QR algorithm. NMIN must be at least 11.

ISPEC=13: (INWIN) Size of the deflation window. This is best set greater than or equal to the number of simultaneous shifts NS. Larger matrices benefit from larger deflation windows.

ISPEC=14: (INIBL) Determines when to stop nibbling and invest in an (expensive) multi-shift QR sweep. If the aggressive early deflation subroutine finds LD converged eigenvalues from an order NW deflation window and LD.GT.(NW*NIBBLE)/100, then the next QR sweep is skipped and early deflation is applied immediately to the remaining active diagonal block. Setting IPARMQ(ISPEC=14) = 0 causes TTQRE to skip a multi-shift QR sweep whenever early deflation finds a converged eigenvalue. Setting IPARMQ(ISPEC=14) greater than or equal to 100 prevents TTQRE from skipping a multi-shift QR sweep.

ISPEC=15: (NSHFTS) The number of simultaneous shifts in a multi-shift QR iteration.

ISPEC=16: (IACC22) IPARMQ is set to 0, 1 or 2 with the following meanings. 0: During the multi-shift QR/QZ sweep, blocked eigenvalue reordering, blocked Hessenberg-triangular reduction, reflections and/or rotations are not accumulated when updating the far-from-diagonal matrix entries. 1: During the multi-shift QR/QZ sweep, blocked eigenvalue reordering, blocked Hessenberg-triangular reduction, reflections and/or rotations are accumulated, and matrix-matrix multiplication is used to update the far-from-diagonal matrix entries. 2: During the multi-shift QR/QZ sweep, blocked eigenvalue reordering, blocked Hessenberg-triangular reduction, reflections and/or rotations are accumulated, and 2-by-2 block structure is exploited during matrix-matrix multiplies. (If xTRMM is slower than xGEMM, then IPARMQ(ISPEC=16)=1 may be more efficient than IPARMQ(ISPEC=16)=2 despite the greater level of arithmetic work implied by the latter choice.)

NAME Input parameter.

NAME is character string

Name of the calling subroutine

OPTS Input parameter.

OPTS is character string

This is a concatenation of the string arguments to TTQRE.

N Input parameter.

N is INTEGER

N is the order of the Hessenberg matrix H.

ILO Input parameter.

ILO is INTEGER

IHI Input parameter.

IHI is INTEGER

It is assumed that H is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N.

LWORK Input parameter.

LWORK is INTEGER

The amount of workspace available.

Related Information

4.18.10 slabad

slabad takes as input the values computed by SLAMCH for underflow and overflow, and returns the square root of each of these values if the log of LARGE is sufficiently large. This subroutine is intended to identify machines with a large exponent range, such as the Crays, and redefine the underflow and overflow limits to be the square roots of the values computed by SLAMCH. This subroutine is needed because SLAMCH does not compensate for poor arithmetic in the upper half of the exponent range, as is found on a Cray.

Syntax

Fortran specification:

```
use armpl_library

subroutine slabad(SMALL, LARGE)
```

C specification:

```
#include "armpl.h"

void slabad_(float *small, float *large);
```

Parameters

SMALL Input and output parameter.

SMALL is REAL

On entry, the underflow threshold as computed by SLAMCH. On exit, if LOG10(LARGE) is sufficiently large, the square root of SMALL, otherwise unchanged.

LARGE Input and output parameter.

LARGE is REAL

On entry, the overflow threshold as computed by SLAMCH. On exit, if LOG10(LARGE) is sufficiently large, the square root of LARGE, otherwise unchanged.

Related Information

For this routine in other precisions, please see *dlabad*.

4.18.11 xerbla

`xerbla` is an error handler for the LAPACK routines. It is called by an LAPACK routine if an input parameter has an invalid value. A message is printed and execution stops.

Installers may consider modifying the STOP statement in order to call system-specific exception-handling facilities.

Syntax

Fortran specification:

```
use armpl_library

subroutine xerbla(SRNAME, INFO)
```

C specification:

```
#include "armpl.h"

void xerbla_(const char *srname, const armpl_int_t *info, ... );
```

Parameters

SRNAME Input parameter.

SRNAME is CHARACTER*(*)

The name of the routine which called XERBLA.

INFO Input parameter.

INFO is INTEGER

The position of the invalid parameter in the parameter list of the calling routine.

Related Information

It also exists with a native C interface as *cblas_xerbla*.

4.18.12 xerbla_array

XERBLA_ARRAY assists other languages in calling XERBLA, the LAPACK and BLAS error handler. Rather than taking a Fortran string argument as the function's name, XERBLA_ARRAY takes an array of single characters along with the array's length. XERBLA_ARRAY then copies up to 32 characters of that array into a Fortran string and passes that to XERBLA. If called with a non-positive SRNAME_LEN, XERBLA_ARRAY will call XERBLA with a string of all blank characters.

Say some macro or other device makes XERBLA_ARRAY available to C99 by a name lapack_xerbla and with a common Fortran calling convention. Then a C99 program could invoke XERBLA via:

```
{
  int flen = strlen(__func__);
  lapack_xerbla(__func__, &amp;flen, &amp;info);
}
```

Providing XERBLA_ARRAY is not necessary for intercepting LAPACK errors. XERBLA_ARRAY calls XERBLA.

Syntax

Fortran specification:

```
use armpl_library

subroutine xerbla_array(SRNAME_ARRAY, SRNAME_LEN, INFO)
```

C specification:

```
#include "armpl.h"

void xerbla_array_(const char *srname_array, const armpl_int_t *srname_len,
                   const armpl_int_t *info, ... );
```

Parameters

SRNAME_ARRAY Input parameter.

SRNAME_ARRAY is CHARACTER(1) array, dimension (SRNAME_LEN)

The name of the routine which called XERBLA_ARRAY.

SRNAME_LEN Input parameter.

SRNAME_LEN is INTEGER

The length of the name in SRNAME_ARRAY.

INFO Input parameter.

INFO is INTEGER

The position of the invalid parameter in the parameter list of the calling routine.

4.19 LAPACKE - C interfaces for LAPACK functions

4.19.1 LAPACKE_cbbcsd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cbbcsd(armpl_int_t matrix_layout, char jobu1, char jobu2,
                           char jobv1t, char jobv2t, char trans,
                           armpl_int_t m, armpl_int_t p, armpl_int_t q,
                           float *theta, float *phi,
                           armpl_singlecomplex_t *u1, armpl_int_t ldu1,
                           armpl_singlecomplex_t *u2, armpl_int_t ldu2,
                           armpl_singlecomplex_t *v1t, armpl_int_t ldv1t,
                           armpl_singlecomplex_t *v2t, armpl_int_t ldv2t,
                           float *b11d, float *b11e, float *b12d, float *b12e,
                           float *b21d, float *b21e, float *b22d,
                           float *b22e);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cbbcsd*, *LAPACKE_dbbcsd*, *LAPACKE_sbbcsd* and *LAPACKE_zbbcsd*. It also exists with a native Fortran interface as *cbbcsd*.

4.19.2 LAPACKE_cbbcsd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cbbcsd_work(armpl_int_t matrix_layout, char jobu1,
                                char jobu2, char jobv1t, char jobv2t,
                                char trans, armpl_int_t m, armpl_int_t p,
                                armpl_int_t q, float *theta, float *phi,
                                armpl_singlecomplex_t *u1, armpl_int_t ldu1,
                                armpl_singlecomplex_t *u2, armpl_int_t ldu2,
                                armpl_singlecomplex_t *v1t, armpl_int_t ldv1t,
                                armpl_singlecomplex_t *v2t, armpl_int_t ldv2t,
                                float *b11d, float *b11e, float *b12d,
                                float *b12e, float *b21d, float *b21e,
                                float *b22d, float *b22e, float *rwork,
                                armpl_int_t lrwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cbbcsd_work*, *LAPACKE_dbbcsd_work*, *LAPACKE_sbbcsd_work* and *LAPACKE_zbbcsd_work*.

4.19.3 LAPACKE_cbdsqr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cbdsqr(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t ncvt, armpl_int_t nru,
                           armpl_int_t ncc, float *d, float *e,
```

(continues on next page)

(continued from previous page)

```
                                armpl_singlecomplex_t *vt, armpl_int_t ldvt,
                                armpl_singlecomplex_t *u, armpl_int_t ldu,
                                armpl_singlecomplex_t *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cbdsqr*, *LAPACKE_dbdsqr*, *LAPACKE_sbdsqr* and *LAPACKE_zbdsqr*. It also exists with a native Fortran interface as *cbdsqr*.

4.19.4 LAPACKE_cbdsqr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cbdsqr_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t ncvt,
                                armpl_int_t nru, armpl_int_t ncc, float *d,
                                float *e, armpl_singlecomplex_t *vt,
                                armpl_int_t ldvt, armpl_singlecomplex_t *u,
                                armpl_int_t ldu, armpl_singlecomplex_t *c,
                                armpl_int_t ldc, float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cbdsqr_work*, *LAPACKE_dbdsqr_work*, *LAPACKE_sbdsqr_work* and *LAPACKE_zbdsqr_work*.

4.19.5 LAPACKE_cgbbrd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgbbrd(armpl_int_t matrix_layout, char vect,
                           armpl_int_t m, armpl_int_t n, armpl_int_t ncc,
                           armpl_int_t kl, armpl_int_t ku,
                           armpl_singlecomplex_t *ab, armpl_int_t ldab,
                           float *d, float *e, armpl_singlecomplex_t *q,
                           armpl_int_t ldq, armpl_singlecomplex_t *pt,
                           armpl_int_t ldpt, armpl_singlecomplex_t *c,
                           armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbbrd*, *LAPACKE_dgbbrd*, *LAPACKE_sgbbrd* and *LAPACKE_zgbbrd*. It also exists with a native Fortran interface as *cgbbrd*.

4.19.6 LAPACKE_cgbbrd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgbbrd_work(armpl_int_t matrix_layout, char vect,
                                armpl_int_t m, armpl_int_t n, armpl_int_t ncc,
                                armpl_int_t kl, armpl_int_t ku,
                                armpl_singlecomplex_t *ab, armpl_int_t ldab,
                                float *d, float *e, armpl_singlecomplex_t *q,
                                armpl_int_t ldq, armpl_singlecomplex_t *pt,
                                armpl_int_t ldpt, armpl_singlecomplex_t *c,
                                armpl_int_t ldc, armpl_singlecomplex_t *work,
                                float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbbrd_work*, *LAPACKE_dgbbrd_work*, *LAPACKE_sgbbrd_work* and *LAPACKE_zgbbrd_work*.

4.19.7 LAPACKE_cgbcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgbcon(armpl_int_t matrix_layout, char norm,
                           armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                           const armpl_singlecomplex_t *ab, armpl_int_t ldab,
                           const armpl_int_t *ipiv, float anorm,
                           float *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbcon*, *LAPACKE_dgbcon*, *LAPACKE_sgbcon* and *LAPACKE_zgbcon*. It also exists with a native Fortran interface as *cgbcon*.

4.19.8 LAPACKE_cgbcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgbcon_work(armpl_int_t matrix_layout, char norm,
                                armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                                const armpl_singlecomplex_t *ab,
                                armpl_int_t ldab, const armpl_int_t *ipiv,
                                float anorm, float *rcond,
                                armpl_singlecomplex_t *work, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbcon_work*, *LAPACKE_dgbcon_work*, *LAPACKE_sgbcon_work* and *LAPACKE_zgbcon_work*.

4.19.9 LAPACKE_cgbequ

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgbequ(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                           const armpl_singlecomplex_t *ab, armpl_int_t ldab,
                           float *r, float *c, float *rowcnd, float *colcnd,
                           float *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbequ*, *LAPACKE_dgbequ*, *LAPACKE_sgbequ* and *LAPACKE_zgbequ*. It also exists with a native Fortran interface as *cgbequ*.

4.19.10 LAPACKE_cgbequ_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgbequ_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                                const armpl_singlecomplex_t *ab,
                                armpl_int_t ldab, float *r, float *c,
                                float *rowcnd, float *colcnd, float *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbequ_work*, *LAPACKE_dgbequ_work*, *LAPACKE_sgbequ_work* and *LAPACKE_zgbequ_work*.

4.19.11 LAPACKE_cgbequb

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgbequb(armpl_int_t matrix_layout, armpl_int_t m,
                            armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                            const armpl_singlecomplex_t *ab, armpl_int_t ldab,
                            float *r, float *c, float *rowcnd, float *colcnd,
                            float *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbequb*, *LAPACKE_dgbequb*, *LAPACKE_sgbequb* and *LAPACKE_zgbequb*. It also exists with a native Fortran interface as *cgbequb*.

4.19.12 LAPACKE_cgbequb_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgbequb_work(armpl_int_t matrix_layout, armpl_int_t m,
                                 armpl_int_t n, armpl_int_t kl,
                                 armpl_int_t ku,
                                 const armpl_singlecomplex_t *ab,
```

(continues on next page)

(continued from previous page)

```
                                armpl_int_t ldab, float *r, float *c,
                                float *rowcnd, float *colcnd, float *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbequb_work*, *LAPACKE_dgbequb_work*, *LAPACKE_sgbequb_work* and *LAPACKE_zgbequb_work*.

4.19.13 LAPACKE_cgbrfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgbrfs(armpl_int_t matrix_layout, char trans,
                           armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                           armpl_int_t nrhs, const armpl_singlecomplex_t *ab,
                           armpl_int_t ldab, const armpl_singlecomplex_t *afb,
                           armpl_int_t ldafb, const armpl_int_t *ipiv,
                           const armpl_singlecomplex_t *b, armpl_int_t ldb,
                           armpl_singlecomplex_t *x, armpl_int_t ldx,
                           float *ferr, float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbrfs*, *LAPACKE_dgbrfs*, *LAPACKE_sgbrfs* and *LAPACKE_zgbrfs*. It also exists with a native Fortran interface as *cgbrfs*.

4.19.14 LAPACKE_cgbrfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgbrfs_work(armpl_int_t matrix_layout, char trans,
                                armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                                armpl_int_t nrhs,
                                const armpl_singlecomplex_t *ab,
                                armpl_int_t ldab,
                                const armpl_singlecomplex_t *afb,
                                armpl_int_t ldafb, const armpl_int_t *ipiv,
                                const armpl_singlecomplex_t *b,
                                armpl_int_t ldb, armpl_singlecomplex_t *x,
                                armpl_int_t ldx, float *ferr, float *berr,
                                armpl_singlecomplex_t *work, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbrfs_work*, *LAPACKE_dgbrfs_work*, *LAPACKE_sgbrfs_work* and *LAPACKE_zgbrfs_work*.

4.19.15 LAPACKE_cgbrfsx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgbrfsx(armpl_int_t matrix_layout, char trans, char equed,
                            armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                            armpl_int_t nrhs, const armpl_singlecomplex_t *ab,
                            armpl_int_t ldab,
                            const armpl_singlecomplex_t *afb,
                            armpl_int_t ldafb, const armpl_int_t *ipiv,
                            const float *r, const float *c,
                            const armpl_singlecomplex_t *b, armpl_int_t ldb,
                            armpl_singlecomplex_t *x, armpl_int_t ldx,
                            float *rcond, float *berr, armpl_int_t n_err_bnds,
                            float *err_bnds_norm, float *err_bnds_comp,
                            armpl_int_t nparams, float *params);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbrfsx*, *LAPACKE_dgbrfsx*, *LAPACKE_sgbrfsx* and *LAPACKE_zgbrfsx*. It also exists with a native Fortran interface as *cgbrfsx*.

4.19.16 LAPACKE_cgbrfsx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgbrfsx_work(armpl_int_t matrix_layout, char trans,
                                 char equed, armpl_int_t n, armpl_int_t kl,
                                 armpl_int_t ku, armpl_int_t nrhs,
                                 const armpl_singlecomplex_t *ab,
                                 armpl_int_t ldab,
                                 const armpl_singlecomplex_t *afb,
                                 armpl_int_t ldafb, const armpl_int_t *ipiv,
                                 const float *r, const float *c,
                                 const armpl_singlecomplex_t *b,
                                 armpl_int_t ldb, armpl_singlecomplex_t *x,
                                 armpl_int_t ldx, float *rcond, float *berr,
                                 armpl_int_t n_err_bnds, float *err_bnds_norm,
                                 float *err_bnds_comp, armpl_int_t nparams,
                                 float *params, armpl_singlecomplex_t *work,
                                 float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbrfsx_work*, *LAPACKE_dgbrfsx_work*, *LAPACKE_sgbrfsx_work* and *LAPACKE_zgbrfsx_work*.

4.19.17 LAPACKE_cgbsv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgbsv(armpl_int_t matrix_layout, armpl_int_t n,
                          armpl_int_t kl, armpl_int_t ku, armpl_int_t nrhs,
                          armpl_singlecomplex_t *ab, armpl_int_t ldab,
                          armpl_int_t *ipiv, armpl_singlecomplex_t *b,
                          armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbsv*, *LAPACKE_dgbsv*, *LAPACKE_sgbsv* and *LAPACKE_zgbsv*. It also exists with a native Fortran interface as *cgbsv*.

4.19.18 LAPACKE_cgbsv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgbsv_work(armpl_int_t matrix_layout, armpl_int_t n,
                               armpl_int_t kl, armpl_int_t ku,
                               armpl_int_t nrhs, armpl_singlecomplex_t *ab,
                               armpl_int_t ldab, armpl_int_t *ipiv,
                               armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbsv_work*, *LAPACKE_dgbsv_work*, *LAPACKE_sgbsv_work* and *LAPACKE_zgbsv_work*.

4.19.19 LAPACKE_cgbsvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgbsvx(armpl_int_t matrix_layout, char fact, char trans,
                           armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                           armpl_int_t nrhs, armpl_singlecomplex_t *ab,
                           armpl_int_t ldab, armpl_singlecomplex_t *afb,
                           armpl_int_t ldafb, armpl_int_t *ipiv, char *equed,
                           float *r, float *c, armpl_singlecomplex_t *b,
                           armpl_int_t ldb, armpl_singlecomplex_t *x,
                           armpl_int_t ldx, float *rcond, float *ferr,
                           float *berr, float *rpivot);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbsvx*, *LAPACKE_dgbsvx*, *LAPACKE_sgbsvx* and *LAPACKE_zgbsvx*. It also exists with a native Fortran interface as *cgbsvx*.

4.19.20 LAPACKE_cgbsvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgbsvx_work(armpl_int_t matrix_layout, char fact,
                                char trans, armpl_int_t n, armpl_int_t kl,
                                armpl_int_t ku, armpl_int_t nrhs,
                                armpl_singlecomplex_t *ab, armpl_int_t ldab,
                                armpl_singlecomplex_t *afb, armpl_int_t ldafb,
                                armpl_int_t *ipiv, char *equed, float *r,
                                float *c, armpl_singlecomplex_t *b,
                                armpl_int_t ldb, armpl_singlecomplex_t *x,
                                armpl_int_t ldx, float *rcond, float *ferr,
                                float *berr, armpl_singlecomplex_t *work,
                                float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbsvx_work*, *LAPACKE_dgbsvx_work*, *LAPACKE_sgbsvx_work* and *LAPACKE_zgbsvx_work*.

4.19.21 LAPACKE_cgbsvxx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgbsvxx(armpl_int_t matrix_layout, char fact, char trans,
                            armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                            armpl_int_t nrhs, armpl_singlecomplex_t *ab,
                            armpl_int_t ldab, armpl_singlecomplex_t *afb,
                            armpl_int_t ldafb, armpl_int_t *ipiv, char *equed,
                            float *r, float *c, armpl_singlecomplex_t *b,
                            armpl_int_t ldb, armpl_singlecomplex_t *x,
                            armpl_int_t ldx, float *rcond, float *rpvgrw,
                            float *berr, armpl_int_t n_err_bnds,
                            float *err_bnds_norm, float *err_bnds_comp,
                            armpl_int_t nparams, float *params);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbsvxx*, *LAPACKE_dgbsvxx*, *LAPACKE_sgbsvxx* and *LAPACKE_zgbsvxx*. It also exists with a native Fortran interface as *cgbsvxx*.

4.19.22 LAPACKE_cgbsvxx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgbsvxx_work(armpl_int_t matrix_layout, char fact,
                                 char trans, armpl_int_t n, armpl_int_t kl,
                                 armpl_int_t ku, armpl_int_t nrhs,
                                 armpl_singlecomplex_t *ab, armpl_int_t ldab,
                                 armpl_singlecomplex_t *afb,
```

(continues on next page)

(continued from previous page)

```
                                armpl_int_t ldafb, armpl_int_t *ipiv,
                                char *equed, float *r, float *c,
                                armpl_singlecomplex_t *b, armpl_int_t ldb,
                                armpl_singlecomplex_t *x, armpl_int_t ldx,
                                float *rcond, float *rpvgrw, float *berr,
                                armpl_int_t n_err_bnds, float *err_bnds_norm,
                                float *err_bnds_comp, armpl_int_t nparams,
                                float *params, armpl_singlecomplex_t *work,
                                float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbsvxx_work*, *LAPACKE_dgbsvxx_work*, *LAPACKE_sgbsvxx_work* and *LAPACKE_zgbsvxx_work*.

4.19.23 LAPACKE_cgbtrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgbtrf(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                           armpl_singlecomplex_t *ab, armpl_int_t ldab,
                           armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbtrf*, *LAPACKE_dgbtrf*, *LAPACKE_sgbtrf* and *LAPACKE_zgbtrf*. It also exists with a native Fortran interface as *cgbtrf*.

4.19.24 LAPACKE_cgbtrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgbtrf_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                                armpl_singlecomplex_t *ab, armpl_int_t ldab,
                                armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbtrf_work*, *LAPACKE_dgbtrf_work*, *LAPACKE_sgbtrf_work* and *LAPACKE_zgbtrf_work*.

4.19.25 LAPACKE_cgbtrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgbtrs(armpl_int_t matrix_layout, char trans,
                           armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                           armpl_int_t nrhs, const armpl_singlecomplex_t *ab,
                           armpl_int_t ldab, const armpl_int_t *ipiv,
                           armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbtrs*, *LAPACKE_dgbtrs*, *LAPACKE_sgbtrs* and *LAPACKE_zgbtrs*. It also exists with a native Fortran interface as *cgbtrs*.

4.19.26 LAPACKE_cgbtrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgbtrs_work(armpl_int_t matrix_layout, char trans,
                                armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                                armpl_int_t nrhs,
                                const armpl_singlecomplex_t *ab,
                                armpl_int_t ldab, const armpl_int_t *ipiv,
                                armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbtrs_work*, *LAPACKE_dgbtrs_work*, *LAPACKE_sgbtrs_work* and *LAPACKE_zgbtrs_work*.

4.19.27 LAPACKE_cgebak

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgebak(armpl_int_t matrix_layout, char job, char side,
                           armpl_int_t n, armpl_int_t ilo, armpl_int_t ihi,
                           const float *scale, armpl_int_t m,
                           armpl_singlecomplex_t *v, armpl_int_t ldv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgebak*, *LAPACKE_dgebak*, *LAPACKE_sgebak* and *LAPACKE_zgebak*. It also exists with a native Fortran interface as *cgebak*.

4.19.28 LAPACKE_cgebak_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgebak_work(armpl_int_t matrix_layout, char job,
                                char side, armpl_int_t n, armpl_int_t ilo,
                                armpl_int_t ihi, const float *scale,
                                armpl_int_t m, armpl_singlecomplex_t *v,
                                armpl_int_t ldv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgebak_work*, *LAPACKE_dgebak_work*, *LAPACKE_sgebak_work* and *LAPACKE_zgebak_work*.

4.19.29 LAPACKE_cgebal

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgebal(armpl_int_t matrix_layout, char job, armpl_int_t n,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           armpl_int_t *ilo, armpl_int_t *ihi, float *scale);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgebal*, *LAPACKE_dgebal*, *LAPACKE_sgebal* and *LAPACKE_zgebal*. It also exists with a native Fortran interface as *cgebal*.

4.19.30 LAPACKE_cgebal_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgebal_work(armpl_int_t matrix_layout, char job,
                                armpl_int_t n, armpl_singlecomplex_t *a,
                                armpl_int_t lda, armpl_int_t *ilo,
                                armpl_int_t *ihi, float *scale);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgebal_work*, *LAPACKE_dgebal_work*, *LAPACKE_sgebal_work* and *LAPACKE_zgebal_work*.

4.19.31 LAPACKE_cgebrd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgebrd(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_singlecomplex_t *a,
                           armpl_int_t lda, float *d, float *e,
                           armpl_singlecomplex_t *tauq,
                           armpl_singlecomplex_t *taup);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.32 LAPACKE_cgebrd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgebrd_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_singlecomplex_t *a,
                                armpl_int_t lda, float *d, float *e,
                                armpl_singlecomplex_t *tauq,
                                armpl_singlecomplex_t *taup,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgebrd_work*, *LAPACKE_dgebrd_work*, *LAPACKE_sgebrd_work* and *LAPACKE_zgebrd_work*.

4.19.33 LAPACKE_cgecon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgecon(armpl_int_t matrix_layout, char norm,
                           armpl_int_t n, const armpl_singlecomplex_t *a,
                           armpl_int_t lda, float anorm, float *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgecon*, *LAPACKE_dgecon*, *LAPACKE_sgecon* and *LAPACKE_zgecon*. It also exists with a native Fortran interface as *cgecon*.

4.19.34 LAPACKE_cgecon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgecon_work(armpl_int_t matrix_layout, char norm,
                                armpl_int_t n, const armpl_singlecomplex_t *a,
                                armpl_int_t lda, float anorm, float *rcond,
                                armpl_singlecomplex_t *work, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgecon_work*, *LAPACKE_dgecon_work*, *LAPACKE_sgecon_work* and *LAPACKE_zgecon_work*.

4.19.35 LAPACKE_cgeequ

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgeequ(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, const armpl_singlecomplex_t *a,
                           armpl_int_t lda, float *r, float *c, float *rowcnd,
                           float *colcnd, float *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeequ*, *LAPACKE_dgeequ*, *LAPACKE_sgeequ* and *LAPACKE_zgeequ*. It also exists with a native Fortran interface as *cgeequ*.

4.19.36 LAPACKE_cgeequ_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgeequ_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, const armpl_singlecomplex_t *a,
                                armpl_int_t lda, float *r, float *c,
                                float *rowcnd, float *colcnd, float *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeequ_work*, *LAPACKE_dgeequ_work*, *LAPACKE_sgeequ_work* and *LAPACKE_zgeequ_work*.

4.19.37 LAPACKE_cgeequb

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgeequb(armpl_int_t matrix_layout, armpl_int_t m,
                            armpl_int_t n, const armpl_singlecomplex_t *a,
                            armpl_int_t lda, float *r, float *c,
                            float *rowcnd, float *colcnd, float *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeequb*, *LAPACKE_dgeequb*, *LAPACKE_sgeequb* and *LAPACKE_zgeequb*. It also exists with a native Fortran interface as *cgeequb*.

4.19.38 LAPACKE_cgeequb_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgeequb_work(armpl_int_t matrix_layout, armpl_int_t m,
                                 armpl_int_t n,
                                 const armpl_singlecomplex_t *a,
                                 armpl_int_t lda, float *r, float *c,
                                 float *rowcnd, float *colcnd, float *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeequb_work*, *LAPACKE_dgeequb_work*, *LAPACKE_sgeequb_work* and *LAPACKE_zgeequb_work*.

4.19.39 LAPACKE_cgees

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgees(armpl_int_t matrix_layout, char jobvs, char sort,
                          LAPACK_C_SELECT1 select, armpl_int_t n,
                          armpl_singlecomplex_t *a, armpl_int_t lda,
                          armpl_int_t *sdim, armpl_singlecomplex_t *w,
                          armpl_singlecomplex_t *vs, armpl_int_t ldvs);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgees*, *LAPACKE_dgees*, *LAPACKE_sgees* and *LAPACKE_zgees*. It also exists with a native Fortran interface as *cgees*.

4.19.40 LAPACKE_cgees_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgees_work(armpl_int_t matrix_layout, char jobvs,
                               char sort, LAPACK_C_SELECT1 select,
                               armpl_int_t n, armpl_singlecomplex_t *a,
                               armpl_int_t lda, armpl_int_t *sdim,
                               armpl_singlecomplex_t *w,
                               armpl_singlecomplex_t *vs, armpl_int_t ldvs,
                               armpl_singlecomplex_t *work, armpl_int_t lwork,
                               float *rwork, armpl_int_t *bwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgees_work*, *LAPACKE_dgees_work*, *LAPACKE_sgees_work* and *LAPACKE_zgees_work*.

4.19.41 LAPACKE_cgeesx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgeesx(armpl_int_t matrix_layout, char jobvs, char sort,
                           LAPACK_C_SELECT1 select, char sense, armpl_int_t n,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           armpl_int_t *sdim, armpl_singlecomplex_t *w,
                           armpl_singlecomplex_t *vs, armpl_int_t ldvs,
                           float *rconde, float *rcondv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeesx*, *LAPACKE_dgeesx*, *LAPACKE_sgeesx* and *LAPACKE_zgeesx*. It also exists with a native Fortran interface as *cgeesx*.

4.19.42 LAPACKE_cgeesx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgeesx_work(armpl_int_t matrix_layout, char jobvs,
                                char sort, LAPACK_C_SELECT1 select,
                                char sense, armpl_int_t n,
                                armpl_singlecomplex_t *a, armpl_int_t lda,
                                armpl_int_t *sdim, armpl_singlecomplex_t *w,
                                armpl_singlecomplex_t *vs, armpl_int_t ldvs,
                                float *rconde, float *rcondv,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork, float *rwork,
                                armpl_int_t *bwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeesx_work*, *LAPACKE_dgeesx_work*, *LAPACKE_sgeesx_work* and *LAPACKE_zgeesx_work*.

4.19.43 LAPACKE_cgeev

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgeev(armpl_int_t matrix_layout, char jobvl, char jobvr,
                          armpl_int_t n, armpl_singlecomplex_t *a,
                          armpl_int_t lda, armpl_singlecomplex_t *w,
                          armpl_singlecomplex_t *vl, armpl_int_t ldvl,
                          armpl_singlecomplex_t *vr, armpl_int_t ldvr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeev*, *LAPACKE_dgeev*, *LAPACKE_sgeev* and *LAPACKE_zgeev*. It also exists with a native Fortran interface as *cgeev*.

4.19.44 LAPACKE_cgeev_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgeev_work(armpl_int_t matrix_layout, char jobvl,
                               char jobvr, armpl_int_t n,
                               armpl_singlecomplex_t *a, armpl_int_t lda,
                               armpl_singlecomplex_t *w,
                               armpl_singlecomplex_t *vl, armpl_int_t ldvl,
                               armpl_singlecomplex_t *vr, armpl_int_t ldvr,
                               armpl_singlecomplex_t *work, armpl_int_t lwork,
                               float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeev_work*, *LAPACKE_dgeev_work*, *LAPACKE_sgeev_work* and *LAPACKE_zgeev_work*.

4.19.45 LAPACKE_cgeevx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgeevx(armpl_int_t matrix_layout, char balanc, char jobvl,
                           char jobvr, char sense, armpl_int_t n,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           armpl_singlecomplex_t *w,
                           armpl_singlecomplex_t *vl, armpl_int_t ldvl,
                           armpl_singlecomplex_t *vr, armpl_int_t ldvr,
                           armpl_int_t *ilo, armpl_int_t *ihi, float *scale,
                           float *abnrm, float *rconde, float *rcondv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeevx*, *LAPACKE_dgeevx*, *LAPACKE_sgeevx* and *LAPACKE_zgeevx*. It also exists with a native Fortran interface as *cgeevx*.

4.19.46 LAPACKE_cgeevx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgeevx_work(armpl_int_t matrix_layout, char balanc,
                                char jobvl, char jobvr, char sense,
                                armpl_int_t n, armpl_singlecomplex_t *a,
                                armpl_int_t lda, armpl_singlecomplex_t *w,
                                armpl_singlecomplex_t *vl, armpl_int_t ldvl,
                                armpl_singlecomplex_t *vr, armpl_int_t ldvr,
                                armpl_int_t *ilo, armpl_int_t *ihi,
                                float *scale, float *abnrm, float *rconde,
                                float *rcondv, armpl_singlecomplex_t *work,
                                armpl_int_t lwork, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeevx_work*, *LAPACKE_dgeevx_work*, *LAPACKE_sgeevx_work* and *LAPACKE_zgeevx_work*.

4.19.47 LAPACKE_cgehrd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgehrd(armpl_int_t matrix_layout, armpl_int_t n,
                           armpl_int_t ilo, armpl_int_t ihi,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           armpl_singlecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgehrd*, *LAPACKE_dgehrd*, *LAPACKE_sgehrd* and *LAPACKE_zgehrd*. It also exists with a native Fortran interface as *cgehrd*.

4.19.48 LAPACKE_cgehrd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgehrd_work(armpl_int_t matrix_layout, armpl_int_t n,
                                armpl_int_t ilo, armpl_int_t ihi,
                                armpl_singlecomplex_t *a, armpl_int_t lda,
                                armpl_singlecomplex_t *tau,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgehrd_work*, *LAPACKE_dgehrd_work*, *LAPACKE_sgehrd_work* and *LAPACKE_zgehrd_work*.

4.19.49 LAPACKE_cgejsv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgejsv(armpl_int_t matrix_layout, char joba, char jobu,
                           char jobv, char jobr, char jobt, char jobp,
                           armpl_int_t m, armpl_int_t n,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           float *sva, armpl_singlecomplex_t *u,
                           armpl_int_t ldu, armpl_singlecomplex_t *v,
                           armpl_int_t ldv, float *stat, armpl_int_t *istat);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgejsv*, *LAPACKE_dgejsv*, *LAPACKE_sgejsv* and *LAPACKE_zgejsv*. It also exists with a native Fortran interface as *cgejsv*.

4.19.50 LAPACKE_cgejsv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgejsv_work(armpl_int_t matrix_layout, char joba,
                                char jobu, char jobv, char jobr, char jobt,
                                char jobp, armpl_int_t m, armpl_int_t n,
                                armpl_singlecomplex_t *a, armpl_int_t lda,
                                float *sva, armpl_singlecomplex_t *u,
                                armpl_int_t ldu, armpl_singlecomplex_t *v,
                                armpl_int_t ldv, armpl_singlecomplex_t *cwork,
                                armpl_int_t lwork, float *work,
                                armpl_int_t lrwork, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgejsv_work*, *LAPACKE_dgejsv_work*, *LAPACKE_sgejsv_work* and *LAPACKE_zgejsv_work*.

4.19.51 LAPACKE_cgelq

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgelq(armpl_int_t matrix_layout, armpl_int_t m,
                          armpl_int_t n, armpl_singlecomplex_t *a,
                          armpl_int_t lda, armpl_singlecomplex_t *t,
                          armpl_int_t tsize);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelq*, *LAPACKE_dgelq*, *LAPACKE_sgelq* and *LAPACKE_zgelq*. It also exists with a native Fortran interface as *cgelq*.

4.19.52 LAPACKE_cgelq2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgelq2(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_singlecomplex_t *a,
                           armpl_int_t lda, armpl_singlecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelq2*, *LAPACKE_dgelq2*, *LAPACKE_sgelq2* and *LAPACKE_zgelq2*. It also exists with a native Fortran interface as *cgelq2*.

4.19.53 LAPACKE_cgelq2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgelq2_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_singlecomplex_t *a,
                                armpl_int_t lda, armpl_singlecomplex_t *tau,
                                armpl_singlecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelq2_work*, *LAPACKE_dgelq2_work*, *LAPACKE_sgelq2_work* and *LAPACKE_zgelq2_work*.

4.19.54 LAPACKE_cgelq_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgelq_work(armpl_int_t matrix_layout, armpl_int_t m,
                               armpl_int_t n, armpl_singlecomplex_t *a,
                               armpl_int_t lda, armpl_singlecomplex_t *t,
                               armpl_int_t tsize, armpl_singlecomplex_t *work,
                               armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelq_work*, *LAPACKE_dgelq_work*, *LAPACKE_sgelq_work* and *LAPACKE_zgelq_work*.

4.19.55 LAPACKE_cgelqf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgelqf(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_singlecomplex_t *a,
                           armpl_int_t lda, armpl_singlecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelqf*, *LAPACKE_dgelqf*, *LAPACKE_sgelqf* and *LAPACKE_zgelqf*. It also exists with a native Fortran interface as *cgelqf*.

4.19.56 LAPACKE_cgelqf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgelqf_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_singlecomplex_t *a,
                                armpl_int_t lda, armpl_singlecomplex_t *tau,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelqf_work*, *LAPACKE_dgelqf_work*, *LAPACKE_sgelqf_work* and *LAPACKE_zgelqf_work*.

4.19.57 LAPACKE_cgels

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgels(armpl_int_t matrix_layout, char trans,
                          armpl_int_t m, armpl_int_t n, armpl_int_t nrhs,
                          armpl_singlecomplex_t *a, armpl_int_t lda,
                          armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgels*, *LAPACKE_dgels*, *LAPACKE_sgels* and *LAPACKE_zgels*. It also exists with a native Fortran interface as *cgels*.

4.19.58 LAPACKE_cgels_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgels_work(armpl_int_t matrix_layout, char trans,
                               armpl_int_t m, armpl_int_t n, armpl_int_t nrhs,
                               armpl_singlecomplex_t *a, armpl_int_t lda,
                               armpl_singlecomplex_t *b, armpl_int_t ldb,
                               armpl_singlecomplex_t *work,
                               armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgels_work*, *LAPACKE_dgels_work*, *LAPACKE_sgels_work* and *LAPACKE_zgels_work*.

4.19.59 LAPACKE_cgelsd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgelsd(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t nrhs,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           armpl_singlecomplex_t *b, armpl_int_t ldb,
                           float *s, float rcond, armpl_int_t *rank);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelsd*, *LAPACKE_dgelsd*, *LAPACKE_sgelsd* and *LAPACKE_zgelsd*. It also exists with a native Fortran interface as *cgelsd*.

4.19.60 LAPACKE_cgelsd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgelsd_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t nrhs,
                                armpl_singlecomplex_t *a, armpl_int_t lda,
                                armpl_singlecomplex_t *b, armpl_int_t ldb,
```

(continues on next page)

(continued from previous page)

```
                                    float *s, float rcond, armpl_int_t *rank,
                                    armpl_singlecomplex_t *work,
                                    armpl_int_t lwork, float *rwork,
                                    armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelsd_work*, *LAPACKE_dgelsd_work*, *LAPACKE_sgelsd_work* and *LAPACKE_zgelsd_work*.

4.19.61 LAPACKE_cgelss

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgelss(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t nrhs,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           armpl_singlecomplex_t *b, armpl_int_t ldb,
                           float *s, float rcond, armpl_int_t *rank);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelss*, *LAPACKE_dgelss*, *LAPACKE_sgelss* and *LAPACKE_zgelss*. It also exists with a native Fortran interface as *cgelss*.

4.19.62 LAPACKE_cgelss_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgelss_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t nrhs,
                                armpl_singlecomplex_t *a, armpl_int_t lda,
                                armpl_singlecomplex_t *b, armpl_int_t ldb,
                                float *s, float rcond, armpl_int_t *rank,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelss_work*, *LAPACKE_dgelss_work*, *LAPACKE_sgelss_work* and *LAPACKE_zgelss_work*.

4.19.63 LAPACKE_cgelsy

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgelsy(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t nrhs,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           armpl_singlecomplex_t *b, armpl_int_t ldb,
                           armpl_int_t *jpvt, float rcond,
                           armpl_int_t *rank);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelsy*, *LAPACKE_dgelsy*, *LAPACKE_sgelsy* and *LAPACKE_zgelsy*. It also exists with a native Fortran interface as *cgelsy*.

4.19.64 LAPACKE_cgelsy_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgelsy_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t nrhs,
                                armpl_singlecomplex_t *a, armpl_int_t lda,
                                armpl_singlecomplex_t *b, armpl_int_t ldb,
                                armpl_int_t *jpvt, float rcond,
                                armpl_int_t *rank,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelsy_work*, *LAPACKE_dgelsy_work*, *LAPACKE_sgelsy_work* and *LAPACKE_zgelsy_work*.

4.19.65 LAPACKE_cgemlq

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgemlq(armpl_int_t matrix_layout, char side, char trans,
                           armpl_int_t m, armpl_int_t n, armpl_int_t k,
                           const armpl_singlecomplex_t *a, armpl_int_t lda,
                           const armpl_singlecomplex_t *t, armpl_int_t tsize,
                           armpl_singlecomplex_t *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgemlq*, *LAPACKE_dgemlq*, *LAPACKE_sgemlq* and *LAPACKE_zgemlq*. It also exists with a native Fortran interface as *cgemlq*.

4.19.66 LAPACKE_cgemlq_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgemlq_work(armpl_int_t matrix_layout, char side,
                                char trans, armpl_int_t m, armpl_int_t n,
                                armpl_int_t k, const armpl_singlecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_singlecomplex_t *t,
                                armpl_int_t tsize, armpl_singlecomplex_t *c,
                                armpl_int_t ldc, armpl_singlecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgemlq_work*, *LAPACKE_dgemlq_work*, *LAPACKE_sgemlq_work* and *LAPACKE_zgemlq_work*.

4.19.67 LAPACKE_cgemqr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgemqr(armpl_int_t matrix_layout, char side, char trans,
                           armpl_int_t m, armpl_int_t n, armpl_int_t k,
                           const armpl_singlecomplex_t *a, armpl_int_t lda,
                           const armpl_singlecomplex_t *t, armpl_int_t tsize,
                           armpl_singlecomplex_t *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgemqr*, *LAPACKE_dgemqr*, *LAPACKE_sgemqr* and *LAPACKE_zgemqr*. It also exists with a native Fortran interface as *cgemqr*.

4.19.68 LAPACKE_cgemqr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgemqr_work(armpl_int_t matrix_layout, char side,
                                char trans, armpl_int_t m, armpl_int_t n,
                                armpl_int_t k, const armpl_singlecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_singlecomplex_t *t,
                                armpl_int_t tsize, armpl_singlecomplex_t *c,
                                armpl_int_t ldc, armpl_singlecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgemqr_work*, *LAPACKE_dgemqr_work*, *LAPACKE_sgemqr_work* and *LAPACKE_zgemqr_work*.

4.19.69 LAPACKE_cgemqrt

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgemqrt(armpl_int_t matrix_layout, char side, char trans,
                            armpl_int_t m, armpl_int_t n, armpl_int_t k,
                            armpl_int_t nb, const armpl_singlecomplex_t *v,
                            armpl_int_t ldv, const armpl_singlecomplex_t *t,
                            armpl_int_t ldt, armpl_singlecomplex_t *c,
                            armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgemqrt*, *LAPACKE_dgemqrt*, *LAPACKE_sgemqrt* and *LAPACKE_zgemqrt*. It also exists with a native Fortran interface as *cgemqrt*.

4.19.70 LAPACKE_cgemqrt_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgemqrt_work(armpl_int_t matrix_layout, char side,
                                 char trans, armpl_int_t m, armpl_int_t n,
                                 armpl_int_t k, armpl_int_t nb,
                                 const armpl_singlecomplex_t *v,
                                 armpl_int_t ldv,
                                 const armpl_singlecomplex_t *t,
                                 armpl_int_t ldt, armpl_singlecomplex_t *c,
                                 armpl_int_t ldc,
                                 armpl_singlecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgemqrt_work*, *LAPACKE_dgemqrt_work*, *LAPACKE_sgemqrt_work* and *LAPACKE_zgemqrt_work*.

4.19.71 LAPACKE_cgeqlf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgeqlf(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_singlecomplex_t *a,
                           armpl_int_t lda, armpl_singlecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqlf*, *LAPACKE_dgeqlf*, *LAPACKE_sgeqlf* and *LAPACKE_zgeqlf*. It also exists with a native Fortran interface as *cgeqlf*.

4.19.72 LAPACKE_cgeqlf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgeqlf_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_singlecomplex_t *a,
                                armpl_int_t lda, armpl_singlecomplex_t *tau,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.73 LAPACKE_cgeqp3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgeqp3(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_singlecomplex_t *a,
                           armpl_int_t lda, armpl_int_t *jpvt,
                           armpl_singlecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqp3*, *LAPACKE_dgeqp3*, *LAPACKE_sgeqp3* and *LAPACKE_zgeqp3*. It also exists with a native Fortran interface as *cgeqp3*.

4.19.74 LAPACKE_cgeqp3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgeqp3_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_singlecomplex_t *a,
                                armpl_int_t lda, armpl_int_t *jpvt,
                                armpl_singlecomplex_t *tau,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqp3_work*, *LAPACKE_dgeqp3_work*, *LAPACKE_sgeqp3_work* and *LAPACKE_zgeqp3_work*.

4.19.75 LAPACKE_cgeqpf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgeqpf(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_singlecomplex_t *a,
                           armpl_int_t lda, armpl_int_t *jpvt,
                           armpl_singlecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqpf*, *LAPACKE_dgeqpf*, *LAPACKE_sgeqpf* and *LAPACKE_zgeqpf*.

4.19.76 LAPACKE_cgeqpf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgeqpf_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_singlecomplex_t *a,
                                armpl_int_t lda, armpl_int_t *jpvt,
                                armpl_singlecomplex_t *tau,
                                armpl_singlecomplex_t *work, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqpf_work*, *LAPACKE_dgeqpf_work*, *LAPACKE_sgeqpf_work* and *LAPACKE_zgeqpf_work*.

4.19.77 LAPACKE_cgeqr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgeqr(armpl_int_t matrix_layout, armpl_int_t m,
                          armpl_int_t n, armpl_singlecomplex_t *a,
                          armpl_int_t lda, armpl_singlecomplex_t *t,
                          armpl_int_t tsize);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqr*, *LAPACKE_dgeqr*, *LAPACKE_sgeqr* and *LAPACKE_zgeqr*. It also exists with a native Fortran interface as *cgeqr*.

4.19.78 LAPACKE_cgeqr2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgeqr2(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_singlecomplex_t *a,
                           armpl_int_t lda, armpl_singlecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqr2*, *LAPACKE_dgeqr2*, *LAPACKE_sgeqr2* and *LAPACKE_zgeqr2*. It also exists with a native Fortran interface as *cgeqr2*.

4.19.79 LAPACKE_cgeqr2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgeqr2_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_singlecomplex_t *a,
                                armpl_int_t lda, armpl_singlecomplex_t *tau,
                                armpl_singlecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqr2_work*, *LAPACKE_dgeqr2_work*, *LAPACKE_sgeqr2_work* and *LAPACKE_zgeqr2_work*.

4.19.80 LAPACKE_cgeqr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgeqr_work(armpl_int_t matrix_layout, armpl_int_t m,
                               armpl_int_t n, armpl_singlecomplex_t *a,
                               armpl_int_t lda, armpl_singlecomplex_t *t,
                               armpl_int_t tsize, armpl_singlecomplex_t *work,
                               armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqr_work*, *LAPACKE_dgeqr_work*, *LAPACKE_sgeqr_work* and *LAPACKE_zgeqr_work*.

4.19.81 LAPACKE_cgeqrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgeqrf(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_singlecomplex_t *a,
                           armpl_int_t lda, armpl_singlecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrf*, *LAPACKE_dgeqrf*, *LAPACKE_sgeqrf* and *LAPACKE_zgeqrf*. It also exists with a native Fortran interface as *cgeqrf*.

4.19.82 LAPACKE_cgeqrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgeqrf_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_singlecomplex_t *a,
                                armpl_int_t lda, armpl_singlecomplex_t *tau,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrf_work*, *LAPACKE_dgeqrf_work*, *LAPACKE_sgeqrf_work* and *LAPACKE_zgeqrf_work*.

4.19.83 LAPACKE_cgeqrfp

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgeqrfp(armpl_int_t matrix_layout, armpl_int_t m,
                            armpl_int_t n, armpl_singlecomplex_t *a,
                            armpl_int_t lda, armpl_singlecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrfp*, *LAPACKE_dgeqrfp*, *LAPACKE_sgeqrfp* and *LAPACKE_zgeqrfp*. It also exists with a native Fortran interface as *cgeqrfp*.

4.19.84 LAPACKE_cgeqrfp_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgeqrfp_work(armpl_int_t matrix_layout, armpl_int_t m,
                                 armpl_int_t n, armpl_singlecomplex_t *a,
                                 armpl_int_t lda, armpl_singlecomplex_t *tau,
                                 armpl_singlecomplex_t *work,
                                 armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrfp_work*, *LAPACKE_dgeqrfp_work*, *LAPACKE_sgeqrfp_work* and *LAPACKE_zgeqrfp_work*.

4.19.85 LAPACKE_cgeqrt

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgeqrt(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t nb,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           armpl_singlecomplex_t *t, armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrt*, *LAPACKE_dgeqrt*, *LAPACKE_sgeqrt* and *LAPACKE_zgeqrt*. It also exists with a native Fortran interface as *cgeqrt*.

4.19.86 LAPACKE_cgeqrt2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgeqrt2(armpl_int_t matrix_layout, armpl_int_t m,
                            armpl_int_t n, armpl_singlecomplex_t *a,
                            armpl_int_t lda, armpl_singlecomplex_t *t,
                            armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrt2*, *LAPACKE_dgeqrt2*, *LAPACKE_sgeqrt2* and *LAPACKE_zgeqrt2*. It also exists with a native Fortran interface as *cgeqrt2*.

4.19.87 LAPACKE_cgeqrt2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgeqrt2_work(armpl_int_t matrix_layout, armpl_int_t m,
                                 armpl_int_t n, armpl_singlecomplex_t *a,
                                 armpl_int_t lda, armpl_singlecomplex_t *t,
                                 armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrt2_work*, *LAPACKE_dgeqrt2_work*, *LAPACKE_sgeqrt2_work* and *LAPACKE_zgeqrt2_work*.

4.19.88 LAPACKE_cgeqrt3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgeqrt3(armpl_int_t matrix_layout, armpl_int_t m,
                            armpl_int_t n, armpl_singlecomplex_t *a,
                            armpl_int_t lda, armpl_singlecomplex_t *t,
                            armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrt3*, *LAPACKE_dgeqrt3*, *LAPACKE_sgeqrt3* and *LAPACKE_zgeqrt3*. It also exists with a native Fortran interface as *cgeqrt3*.

4.19.89 LAPACKE_cgeqrt3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgeqrt3_work(armpl_int_t matrix_layout, armpl_int_t m,
                                 armpl_int_t n, armpl_singlecomplex_t *a,
                                 armpl_int_t lda, armpl_singlecomplex_t *t,
                                 armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrt3_work*, *LAPACKE_dgeqrt3_work*, *LAPACKE_sgeqrt3_work* and *LAPACKE_zgeqrt3_work*.

4.19.90 LAPACKE_cgeqrt_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgeqrt_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t nb,
                                armpl_singlecomplex_t *a, armpl_int_t lda,
                                armpl_singlecomplex_t *t, armpl_int_t ldt,
                                armpl_singlecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrt_work*, *LAPACKE_dgeqrt_work*, *LAPACKE_sgeqrt_work* and *LAPACKE_zgeqrt_work*.

4.19.91 LAPACKE_cgerfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgerfs(armpl_int_t matrix_layout, char trans,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_singlecomplex_t *a, armpl_int_t lda,
                           const armpl_singlecomplex_t *af, armpl_int_t ldaf,
                           const armpl_int_t *ipiv,
                           const armpl_singlecomplex_t *b, armpl_int_t ldb,
                           armpl_singlecomplex_t *x, armpl_int_t ldx,
                           float *ferr, float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgerfs*, *LAPACKE_dgerfs*, *LAPACKE_sgerfs* and *LAPACKE_zgerfs*. It also exists with a native Fortran interface as *cgerfs*.

4.19.92 LAPACKE_cgerfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgerfs_work(armpl_int_t matrix_layout, char trans,
                                armpl_int_t n, armpl_int_t nrhs,
                                const armpl_singlecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_singlecomplex_t *af,
                                armpl_int_t ldaf, const armpl_int_t *ipiv,
                                const armpl_singlecomplex_t *b,
                                armpl_int_t ldb, armpl_singlecomplex_t *x,
                                armpl_int_t ldx, float *ferr, float *berr,
                                armpl_singlecomplex_t *work, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgerfs_work*, *LAPACKE_dgerfs_work*, *LAPACKE_sgerfs_work* and *LAPACKE_zgerfs_work*.

4.19.93 LAPACKE_cgerfsx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgerfsx(armpl_int_t matrix_layout, char trans, char equed,
                            armpl_int_t n, armpl_int_t nrhs,
                            const armpl_singlecomplex_t *a, armpl_int_t lda,
                            const armpl_singlecomplex_t *af, armpl_int_t ldaf,
                            const armpl_int_t *ipiv, const float *r,
                            const float *c, const armpl_singlecomplex_t *b,
                            armpl_int_t ldb, armpl_singlecomplex_t *x,
                            armpl_int_t ldx, float *rcond, float *berr,
                            armpl_int_t n_err_bnds, float *err_bnds_norm,
                            float *err_bnds_comp, armpl_int_t nparams,
                            float *params);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgerfsx*, *LAPACKE_dgerfsx*, *LAPACKE_sgerfsx* and *LAPACKE_zgerfsx*. It also exists with a native Fortran interface as *cgerfsx*.

4.19.94 LAPACKE_cgerfsx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgerfsx_work(armpl_int_t matrix_layout, char trans,
                                 char equed, armpl_int_t n, armpl_int_t nrhs,
                                 const armpl_singlecomplex_t *a,
                                 armpl_int_t lda,
                                 const armpl_singlecomplex_t *af,
                                 armpl_int_t ldaf, const armpl_int_t *ipiv,
                                 const float *r, const float *c,
                                 const armpl_singlecomplex_t *b,
                                 armpl_int_t ldb, armpl_singlecomplex_t *x,
                                 armpl_int_t ldx, float *rcond, float *berr,
                                 armpl_int_t n_err_bnds, float *err_bnds_norm,
                                 float *err_bnds_comp, armpl_int_t nparams,
                                 float *params, armpl_singlecomplex_t *work,
                                 float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgerfsx_work*, *LAPACKE_dgerfsx_work*, *LAPACKE_sgerfsx_work* and *LAPACKE_zgerfsx_work*.

4.19.95 LAPACKE_cgerqf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgerqf(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_singlecomplex_t *a,
                           armpl_int_t lda, armpl_singlecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgerqf*, *LAPACKE_dgerqf*, *LAPACKE_sgerqf* and *LAPACKE_zgerqf*. It also exists with a native Fortran interface as *cgerqf*.

4.19.96 LAPACKE_cgerqf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgerqf_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_singlecomplex_t *a,
                                armpl_int_t lda, armpl_singlecomplex_t *tau,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgerqf_work*, *LAPACKE_dgerqf_work*, *LAPACKE_sgerqf_work* and *LAPACKE_zgerqf_work*.

4.19.97 LAPACKE_cgesdd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgesdd(armpl_int_t matrix_layout, char jobz,
                           armpl_int_t m, armpl_int_t n,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           float *s, armpl_singlecomplex_t *u,
                           armpl_int_t ldu, armpl_singlecomplex_t *vt,
                           armpl_int_t ldvt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesdd*, *LAPACKE_dgesdd*, *LAPACKE_sgesdd* and *LAPACKE_zgesdd*. It also exists with a native Fortran interface as *cgesdd*.

4.19.98 LAPACKE_cgesdd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgesdd_work(armpl_int_t matrix_layout, char jobz,
                                armpl_int_t m, armpl_int_t n,
                                armpl_singlecomplex_t *a, armpl_int_t lda,
                                float *s, armpl_singlecomplex_t *u,
                                armpl_int_t ldu, armpl_singlecomplex_t *vt,
                                armpl_int_t ldvt, armpl_singlecomplex_t *work,
                                armpl_int_t lwork, float *rwork,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesdd_work*, *LAPACKE_dgesdd_work*, *LAPACKE_sgesdd_work* and *LAPACKE_zgesdd_work*.

4.19.99 LAPACKE_cgesv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgesv(armpl_int_t matrix_layout, armpl_int_t n,
                          armpl_int_t nrhs, armpl_singlecomplex_t *a,
                          armpl_int_t lda, armpl_int_t *ipiv,
                          armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesv*, *LAPACKE_dgesv*, *LAPACKE_sgesv* and *LAPACKE_zgesv*. It also exists with a native Fortran interface as *cgesv*.

4.19.100 LAPACKE_cgesv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgesv_work(armpl_int_t matrix_layout, armpl_int_t n,
                               armpl_int_t nrhs, armpl_singlecomplex_t *a,
                               armpl_int_t lda, armpl_int_t *ipiv,
                               armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesv_work*, *LAPACKE_dgesv_work*, *LAPACKE_sgesv_work* and *LAPACKE_zgesv_work*.

4.19.101 LAPACKE_cgesvd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgesvd(armpl_int_t matrix_layout, char jobu, char jobvt,
                           armpl_int_t m, armpl_int_t n,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           float *s, armpl_singlecomplex_t *u,
                           armpl_int_t ldu, armpl_singlecomplex_t *vt,
                           armpl_int_t ldvt, float *superb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvd*, *LAPACKE_dgesvd*, *LAPACKE_sgesvd* and *LAPACKE_zgesvd*. It also exists with a native Fortran interface as *cgesvd*.

4.19.102 LAPACKE_cgesvd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgesvd_work(armpl_int_t matrix_layout, char jobu,
                                char jobvt, armpl_int_t m, armpl_int_t n,
                                armpl_singlecomplex_t *a, armpl_int_t lda,
                                float *s, armpl_singlecomplex_t *u,
                                armpl_int_t ldu, armpl_singlecomplex_t *vt,
                                armpl_int_t ldvt, armpl_singlecomplex_t *work,
                                armpl_int_t lwork, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvd_work*, *LAPACKE_dgesvd_work*, *LAPACKE_sgesvd_work* and *LAPACKE_zgesvd_work*.

4.19.103 LAPACKE_cgesvdx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgesvdx(armpl_int_t matrix_layout, char jobu, char jobvt,
                            char range, armpl_int_t m, armpl_int_t n,
```

(continues on next page)

(continued from previous page)

```
                                armpl_singlecomplex_t *a, armpl_int_t lda,
                                float vl, float vu, armpl_int_t il,
                                armpl_int_t iu, armpl_int_t *ns, float *s,
                                armpl_singlecomplex_t *u, armpl_int_t ldu,
                                armpl_singlecomplex_t *vt, armpl_int_t ldvt,
                                armpl_int_t *superb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvdx*, *LAPACKE_dgesvdx*, *LAPACKE_sgesvdx* and *LAPACKE_zgesvdx*. It also exists with a native Fortran interface as *cgesvdx*.

4.19.104 LAPACKE_cgesvdx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgesvdx_work(armpl_int_t matrix_layout, char jobu,
                                 char jobvt, char range, armpl_int_t m,
                                 armpl_int_t n, armpl_singlecomplex_t *a,
                                 armpl_int_t lda, float vl, float vu,
                                 armpl_int_t il, armpl_int_t iu,
                                 armpl_int_t *ns, float *s,
                                 armpl_singlecomplex_t *u, armpl_int_t ldu,
                                 armpl_singlecomplex_t *vt, armpl_int_t ldvt,
                                 armpl_singlecomplex_t *work,
                                 armpl_int_t lwork, float *rwork,
                                 armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvdx_work*, *LAPACKE_dgesvdx_work*, *LAPACKE_sgesvdx_work* and *LAPACKE_zgesvdx_work*.

4.19.105 LAPACKE_cgesvj

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgesvj(armpl_int_t matrix_layout, char joba, char jobu,
                           char jobv, armpl_int_t m, armpl_int_t n,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           float *sva, armpl_int_t mv,
                           armpl_singlecomplex_t *v, armpl_int_t ldv,
                           float *stat);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvj*, *LAPACKE_dgesvj*, *LAPACKE_sgesvj* and *LAPACKE_zgesvj*. It also exists with a native Fortran interface as *cgesvj*.

4.19.106 LAPACKE_cgesvj_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgesvj_work(armpl_int_t matrix_layout, char joba,
                                char jobu, char jobv, armpl_int_t m,
                                armpl_int_t n, armpl_singlecomplex_t *a,
                                armpl_int_t lda, float *sva, armpl_int_t mv,
                                armpl_singlecomplex_t *v, armpl_int_t ldv,
                                armpl_singlecomplex_t *cwork,
                                armpl_int_t lwork, float *rwork,
                                armpl_int_t lrwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvj_work*, *LAPACKE_dgesvj_work*, *LAPACKE_sgesvj_work* and *LAPACKE_zgesvj_work*.

4.19.107 LAPACKE_cgesvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgesvx(armpl_int_t matrix_layout, char fact, char trans,
                           armpl_int_t n, armpl_int_t nrhs,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           armpl_singlecomplex_t *af, armpl_int_t ldaf,
                           armpl_int_t *ipiv, char *equed, float *r, float *c,
                           armpl_singlecomplex_t *b, armpl_int_t ldb,
                           armpl_singlecomplex_t *x, armpl_int_t ldx,
                           float *rcond, float *ferr, float *berr,
                           float *rpivot);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvx*, *LAPACKE_dgesvx*, *LAPACKE_sgesvx* and *LAPACKE_zgesvx*. It also exists with a native Fortran interface as *cgesvx*.

4.19.108 LAPACKE_cgesvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgesvx_work(armpl_int_t matrix_layout, char fact,
                                char trans, armpl_int_t n, armpl_int_t nrhs,
                                armpl_singlecomplex_t *a, armpl_int_t lda,
                                armpl_singlecomplex_t *af, armpl_int_t ldaf,
                                armpl_int_t *ipiv, char *equed, float *r,
                                float *c, armpl_singlecomplex_t *b,
```

(continues on next page)

(continued from previous page)

```
                                armpl_int_t ldb, armpl_singlecomplex_t *x,
                                armpl_int_t ldx, float *rcond, float *ferr,
                                float *berr, armpl_singlecomplex_t *work,
                                float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvx_work*, *LAPACKE_dgesvx_work*, *LAPACKE_sgesvx_work* and *LAPACKE_zgesvx_work*.

4.19.109 LAPACKE_cgesvxx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgesvxx(armpl_int_t matrix_layout, char fact, char trans,
                            armpl_int_t n, armpl_int_t nrhs,
                            armpl_singlecomplex_t *a, armpl_int_t lda,
                            armpl_singlecomplex_t *af, armpl_int_t ldaf,
                            armpl_int_t *ipiv, char *equed, float *r,
                            float *c, armpl_singlecomplex_t *b,
                            armpl_int_t ldb, armpl_singlecomplex_t *x,
                            armpl_int_t ldx, float *rcond, float *rpvgrw,
                            float *berr, armpl_int_t n_err_bnds,
                            float *err_bnds_norm, float *err_bnds_comp,
                            armpl_int_t nparams, float *params);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvxx*, *LAPACKE_dgesvxx*, *LAPACKE_sgesvxx* and *LAPACKE_zgesvxx*. It also exists with a native Fortran interface as *cgesvxx*.

4.19.110 LAPACKE_cgesvxx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgesvxx_work(armpl_int_t matrix_layout, char fact,
                                 char trans, armpl_int_t n, armpl_int_t nrhs,
                                 armpl_singlecomplex_t *a, armpl_int_t lda,
                                 armpl_singlecomplex_t *af, armpl_int_t ldaf,
                                 armpl_int_t *ipiv, char *equed, float *r,
                                 float *c, armpl_singlecomplex_t *b,
                                 armpl_int_t ldb, armpl_singlecomplex_t *x,
                                 armpl_int_t ldx, float *rcond, float *rpvgrw,
                                 float *berr, armpl_int_t n_err_bnds,
                                 float *err_bnds_norm, float *err_bnds_comp,
                                 armpl_int_t nparams, float *params,
                                 armpl_singlecomplex_t *work, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvxx_work*, *LAPACKE_dgesvxx_work*, *LAPACKE_sgesvxx_work* and *LAPACKE_zgesvxx_work*.

4.19.111 LAPACKE_cgetf2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgetf2(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_singlecomplex_t *a,
                           armpl_int_t lda, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetf2*, *LAPACKE_dgetf2*, *LAPACKE_sgetf2* and *LAPACKE_zgetf2*. It also exists with a native Fortran interface as *cgetf2*.

4.19.112 LAPACKE_cgetf2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgetf2_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_singlecomplex_t *a,
                                armpl_int_t lda, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetf2_work*, *LAPACKE_dgetf2_work*, *LAPACKE_sgetf2_work* and *LAPACKE_zgetf2_work*.

4.19.113 LAPACKE_cgetrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgetrf(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_singlecomplex_t *a,
                           armpl_int_t lda, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetrf*, *LAPACKE_dgetrf*, *LAPACKE_sgetrf* and *LAPACKE_zgetrf*. It also exists with a native Fortran interface as *cgetrf*.

4.19.114 LAPACKE_cgetrf2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgetrf2(armpl_int_t matrix_layout, armpl_int_t m,
                            armpl_int_t n, armpl_singlecomplex_t *a,
                            armpl_int_t lda, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetrf2*, *LAPACKE_dgetrf2*, *LAPACKE_sgetrf2* and *LAPACKE_zgetrf2*. It also exists with a native Fortran interface as *cgetrf2*.

4.19.115 LAPACKE_cgetrf2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgetrf2_work(armpl_int_t matrix_layout, armpl_int_t m,
                                 armpl_int_t n, armpl_singlecomplex_t *a,
                                 armpl_int_t lda, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetrf2_work*, *LAPACKE_dgetrf2_work*, *LAPACKE_sgetrf2_work* and *LAPACKE_zgetrf2_work*.

4.19.116 LAPACKE_cgetrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgetrf_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_singlecomplex_t *a,
                                armpl_int_t lda, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetrf_work*, *LAPACKE_dgetrf_work*, *LAPACKE_sgetrf_work* and *LAPACKE_zgetrf_work*.

4.19.117 LAPACKE_cgetri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgetri(armpl_int_t matrix_layout, armpl_int_t n,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           const armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetri*, *LAPACKE_dgetri*, *LAPACKE_sgetri* and *LAPACKE_zgetri*. It also exists with a native Fortran interface as *cgetri*.

4.19.118 LAPACKE_cgetri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgetri_work(armpl_int_t matrix_layout, armpl_int_t n,
                                armpl_singlecomplex_t *a, armpl_int_t lda,
                                const armpl_int_t *ipiv,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetri_work*, *LAPACKE_dgetri_work*, *LAPACKE_sgetri_work* and *LAPACKE_zgetri_work*.

4.19.119 LAPACKE_cgetrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgetrs(armpl_int_t matrix_layout, char trans,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_singlecomplex_t *a, armpl_int_t lda,
                           const armpl_int_t *ipiv, armpl_singlecomplex_t *b,
                           armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetrs*, *LAPACKE_dgetrs*, *LAPACKE_sgetrs* and *LAPACKE_zgetrs*. It also exists with a native Fortran interface as *cgetrs*.

4.19.120 LAPACKE_cgetrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgetrs_work(armpl_int_t matrix_layout, char trans,
                                armpl_int_t n, armpl_int_t nrhs,
                                const armpl_singlecomplex_t *a,
                                armpl_int_t lda, const armpl_int_t *ipiv,
                                armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetrs_work*, *LAPACKE_dgetrs_work*, *LAPACKE_sgetrs_work* and *LAPACKE_zgetrs_work*.

4.19.121 LAPACKE_cgetsls

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgetsls(armpl_int_t matrix_layout, char trans,
                            armpl_int_t m, armpl_int_t n, armpl_int_t nrhs,
                            armpl_singlecomplex_t *a, armpl_int_t lda,
                            armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetsls*, *LAPACKE_dgetsls*, *LAPACKE_sgetsls* and *LAPACKE_zgetsls*. It also exists with a native Fortran interface as *cgetsls*.

4.19.122 LAPACKE_cgetsls_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgetsls_work(armpl_int_t matrix_layout, char trans,
                                 armpl_int_t m, armpl_int_t n,
                                 armpl_int_t nrhs, armpl_singlecomplex_t *a,
                                 armpl_int_t lda, armpl_singlecomplex_t *b,
                                 armpl_int_t ldb, armpl_singlecomplex_t *work,
                                 armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetsls_work*, *LAPACKE_dgetsls_work*, *LAPACKE_sgetsls_work* and *LAPACKE_zgetsls_work*.

4.19.123 LAPACKE_cggbak

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cggbak(armpl_int_t matrix_layout, char job, char side,
                           armpl_int_t n, armpl_int_t ilo, armpl_int_t ihi,
                           const float *lscale, const float *rscale,
                           armpl_int_t m, armpl_singlecomplex_t *v,
                           armpl_int_t ldv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggbak*, *LAPACKE_dggbak*, *LAPACKE_sggbak* and *LAPACKE_zggbak*. It also exists with a native Fortran interface as *cggbak*.

4.19.124 LAPACKE_cggbak_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cggbak_work(armpl_int_t matrix_layout, char job,
                                char side, armpl_int_t n, armpl_int_t ilo,
                                armpl_int_t ihi, const float *lscale,
                                const float *rscale, armpl_int_t m,
                                armpl_singlecomplex_t *v, armpl_int_t ldv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggbak_work*, *LAPACKE_dggbak_work*, *LAPACKE_sggbak_work* and *LAPACKE_zggbak_work*.

4.19.125 LAPACKE_cggbal

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cggbal(armpl_int_t matrix_layout, char job, armpl_int_t n,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           armpl_singlecomplex_t *b, armpl_int_t ldb,
                           armpl_int_t *ilo, armpl_int_t *ihi, float *lscale,
                           float *rscale);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggbal*, *LAPACKE_dggbal*, *LAPACKE_sggbal* and *LAPACKE_zggbal*. It also exists with a native Fortran interface as *cggbal*.

4.19.126 LAPACKE_cggbal_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cggbal_work(armpl_int_t matrix_layout, char job,
                                armpl_int_t n, armpl_singlecomplex_t *a,
                                armpl_int_t lda, armpl_singlecomplex_t *b,
                                armpl_int_t ldb, armpl_int_t *ilo,
                                armpl_int_t *ihi, float *lscale,
                                float *rscale, float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggbal_work*, *LAPACKE_dggbal_work*, *LAPACKE_sggbal_work* and *LAPACKE_zggbal_work*.

4.19.127 LAPACKE_cgges

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgges(armpl_int_t matrix_layout, char jobvsl, char jobvsr,
                          char sort, LAPACK_C_SELECT2 selctg, armpl_int_t n,
                          armpl_singlecomplex_t *a, armpl_int_t lda,
                          armpl_singlecomplex_t *b, armpl_int_t ldb,
                          armpl_int_t *sdim, armpl_singlecomplex_t *alpha,
                          armpl_singlecomplex_t *beta,
                          armpl_singlecomplex_t *vsl, armpl_int_t ldvsl,
                          armpl_singlecomplex_t *vsr, armpl_int_t ldvsr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgges*, *LAPACKE_dgges*, *LAPACKE_sgges* and *LAPACKE_zgges*. It also exists with a native Fortran interface as *cgges*.

4.19.128 LAPACKE_cgges3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgges3(armpl_int_t matrix_layout, char jobvsl,
                           char jobvsr, char sort, LAPACK_C_SELECT2 selctg,
                           armpl_int_t n, armpl_singlecomplex_t *a,
                           armpl_int_t lda, armpl_singlecomplex_t *b,
                           armpl_int_t ldb, armpl_int_t *sdim,
                           armpl_singlecomplex_t *alpha,
                           armpl_singlecomplex_t *beta,
                           armpl_singlecomplex_t *vsl, armpl_int_t ldvsl,
                           armpl_singlecomplex_t *vsr, armpl_int_t ldvsr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgges3*, *LAPACKE_dgges3*, *LAPACKE_sgges3* and *LAPACKE_zgges3*. It also exists with a native Fortran interface as *cgges3*.

4.19.129 LAPACKE_cgges3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgges3_work(armpl_int_t matrix_layout, char jobvsl,
                                char jobvsr, char sort,
                                LAPACK_C_SELECT2 selctg, armpl_int_t n,
                                armpl_singlecomplex_t *a, armpl_int_t lda,
                                armpl_singlecomplex_t *b, armpl_int_t ldb,
                                armpl_int_t *sdim,
                                armpl_singlecomplex_t *alpha,
                                armpl_singlecomplex_t *beta,
                                armpl_singlecomplex_t *vsl, armpl_int_t ldvsl,
                                armpl_singlecomplex_t *vsr, armpl_int_t ldvsr,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork, float *rwork,
                                armpl_int_t *bwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgges3_work*, *LAPACKE_dgges3_work*, *LAPACKE_sgges3_work* and *LAPACKE_zgges3_work*.

4.19.130 LAPACKE_cgges_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgges_work(armpl_int_t matrix_layout, char jobvsl,
                               char jobvsr, char sort,
                               LAPACK_C_SELECT2 selctg, armpl_int_t n,
                               armpl_singlecomplex_t *a, armpl_int_t lda,
                               armpl_singlecomplex_t *b, armpl_int_t ldb,
                               armpl_int_t *sdim,
                               armpl_singlecomplex_t *alpha,
                               armpl_singlecomplex_t *beta,
                               armpl_singlecomplex_t *vsl, armpl_int_t ldvsl,
                               armpl_singlecomplex_t *vsr, armpl_int_t ldvsr,
                               armpl_singlecomplex_t *work, armpl_int_t lwork,
                               float *rwork, armpl_int_t *bwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgges_work*, *LAPACKE_dgges_work*, *LAPACKE_sgges_work* and *LAPACKE_zgges_work*.

4.19.131 LAPACKE_cggesx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cggesx(armpl_int_t matrix_layout, char jobvsl,
                           char jobvsr, char sort, LAPACK_C_SELECT2 selctg,
                           char sense, armpl_int_t n,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           armpl_singlecomplex_t *b, armpl_int_t ldb,
                           armpl_int_t *sdim, armpl_singlecomplex_t *alpha,
                           armpl_singlecomplex_t *beta,
                           armpl_singlecomplex_t *vsl, armpl_int_t ldvsl,
                           armpl_singlecomplex_t *vsr, armpl_int_t ldvsr,
                           float *rconde, float *rcondv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggesx*, *LAPACKE_dggesx*, *LAPACKE_sggesx* and *LAPACKE_zggesx*. It also exists with a native Fortran interface as *cggesx*.

4.19.132 LAPACKE_cggesx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cggesx_work(armpl_int_t matrix_layout, char jobvsl,
                                char jobvsr, char sort,
                                LAPACK_C_SELECT2 selctg, char sense,
                                armpl_int_t n, armpl_singlecomplex_t *a,
                                armpl_int_t lda, armpl_singlecomplex_t *b,
```

(continues on next page)

(continued from previous page)

```
                                armpl_int_t ldb, armpl_int_t *sdim,
                                armpl_singlecomplex_t *alpha,
                                armpl_singlecomplex_t *beta,
                                armpl_singlecomplex_t *vsl, armpl_int_t ldvsl,
                                armpl_singlecomplex_t *vsr, armpl_int_t ldvsr,
                                float *rconde, float *rcondv,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork, float *rwork,
                                armpl_int_t *iwork, armpl_int_t liwork,
                                armpl_int_t *bwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggesx_work*, *LAPACKE_dggesx_work*, *LAPACKE_sggesx_work* and *LAPACKE_zggesx_work*.

4.19.133 LAPACKE_cggev

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cggev(armpl_int_t matrix_layout, char jobvl, char jobvr,
                          armpl_int_t n, armpl_singlecomplex_t *a,
                          armpl_int_t lda, armpl_singlecomplex_t *b,
                          armpl_int_t ldb, armpl_singlecomplex_t *alpha,
                          armpl_singlecomplex_t *beta,
                          armpl_singlecomplex_t *vl, armpl_int_t ldvl,
                          armpl_singlecomplex_t *vr, armpl_int_t ldvr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggev*, *LAPACKE_dggev*, *LAPACKE_sggev* and *LAPACKE_zggev*. It also exists with a native Fortran interface as *cggev*.

4.19.134 LAPACKE_cggev3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cggev3(armpl_int_t matrix_layout, char jobvl, char jobvr,
                           armpl_int_t n, armpl_singlecomplex_t *a,
                           armpl_int_t lda, armpl_singlecomplex_t *b,
                           armpl_int_t ldb, armpl_singlecomplex_t *alpha,
                           armpl_singlecomplex_t *beta,
                           armpl_singlecomplex_t *vl, armpl_int_t ldvl,
                           armpl_singlecomplex_t *vr, armpl_int_t ldvr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggev3*, *LAPACKE_dggev3*, *LAPACKE_sggev3* and *LAPACKE_zggev3*. It also exists with a native Fortran interface as *cggev3*.

4.19.135 LAPACKE_cggev3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cggev3_work(armpl_int_t matrix_layout, char jobvl,
                                char jobvr, armpl_int_t n,
                                armpl_singlecomplex_t *a, armpl_int_t lda,
                                armpl_singlecomplex_t *b, armpl_int_t ldb,
                                armpl_singlecomplex_t *alpha,
                                armpl_singlecomplex_t *beta,
                                armpl_singlecomplex_t *vl, armpl_int_t ldvl,
                                armpl_singlecomplex_t *vr, armpl_int_t ldvr,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggev3_work*, *LAPACKE_dggev3_work*, *LAPACKE_sggev3_work* and *LAPACKE_zggev3_work*.

4.19.136 LAPACKE_cggev_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cggev_work(armpl_int_t matrix_layout, char jobvl,
                               char jobvr, armpl_int_t n,
                               armpl_singlecomplex_t *a, armpl_int_t lda,
                               armpl_singlecomplex_t *b, armpl_int_t ldb,
                               armpl_singlecomplex_t *alpha,
                               armpl_singlecomplex_t *beta,
                               armpl_singlecomplex_t *vl, armpl_int_t ldvl,
                               armpl_singlecomplex_t *vr, armpl_int_t ldvr,
                               armpl_singlecomplex_t *work, armpl_int_t lwork,
                               float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggev_work*, *LAPACKE_dggev_work*, *LAPACKE_sggev_work* and *LAPACKE_zggev_work*.

4.19.137 LAPACKE_cggevx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cggevx(armpl_int_t matrix_layout, char balanc, char jobvl,
                           char jobvr, char sense, armpl_int_t n,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           armpl_singlecomplex_t *b, armpl_int_t ldb,
                           armpl_singlecomplex_t *alpha,
                           armpl_singlecomplex_t *beta,
                           armpl_singlecomplex_t *vl, armpl_int_t ldvl,
                           armpl_singlecomplex_t *vr, armpl_int_t ldvr,
                           armpl_int_t *ilo, armpl_int_t *ihi, float *lscale,
                           float *rscale, float *abnrm, float *bbnrm,
                           float *rconde, float *rcondv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggevx*, *LAPACKE_dggevx*, *LAPACKE_sggevx* and *LAPACKE_zggevx*. It also exists with a native Fortran interface as *cggevx*.

4.19.138 LAPACKE_cggevx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cggevx_work(armpl_int_t matrix_layout, char balanc,
                                char jobvl, char jobvr, char sense,
                                armpl_int_t n, armpl_singlecomplex_t *a,
                                armpl_int_t lda, armpl_singlecomplex_t *b,
                                armpl_int_t ldb, armpl_singlecomplex_t *alpha,
                                armpl_singlecomplex_t *beta,
                                armpl_singlecomplex_t *vl, armpl_int_t ldvl,
                                armpl_singlecomplex_t *vr, armpl_int_t ldvr,
                                armpl_int_t *ilo, armpl_int_t *ihi,
                                float *lscale, float *rscale, float *abnrm,
                                float *bbnrm, float *rconde, float *rcondv,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork, float *rwork,
                                armpl_int_t *iwork, armpl_int_t *bwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggevx_work*, *LAPACKE_dggevx_work*, *LAPACKE_sggevx_work* and *LAPACKE_zggevx_work*.

4.19.139 LAPACKE_cggglm

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cggglm(armpl_int_t matrix_layout, armpl_int_t n,
                           armpl_int_t m, armpl_int_t p,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           armpl_singlecomplex_t *b, armpl_int_t ldb,
                           armpl_singlecomplex_t *d, armpl_singlecomplex_t *x,
                           armpl_singlecomplex_t *y);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggglm*, *LAPACKE_dggglm*, *LAPACKE_sggglm* and *LAPACKE_zggglm*. It also exists with a native Fortran interface as *cggglm*.

4.19.140 LAPACKE_cggglm_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cggglm_work(armpl_int_t matrix_layout, armpl_int_t n,
                                armpl_int_t m, armpl_int_t p,
                                armpl_singlecomplex_t *a, armpl_int_t lda,
                                armpl_singlecomplex_t *b, armpl_int_t ldb,
                                armpl_singlecomplex_t *d,
                                armpl_singlecomplex_t *x,
                                armpl_singlecomplex_t *y,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggglm_work*, *LAPACKE_dggglm_work*, *LAPACKE_sggglm_work* and *LAPACKE_zggglm_work*.

4.19.141 LAPACKE_cgghd3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgghd3(armpl_int_t matrix_layout, char compq, char compz,
                           armpl_int_t n, armpl_int_t ilo, armpl_int_t ihi,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           armpl_singlecomplex_t *b, armpl_int_t ldb,
                           armpl_singlecomplex_t *q, armpl_int_t ldq,
                           armpl_singlecomplex_t *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgghd3*, *LAPACKE_dgghd3*, *LAPACKE_sgghd3* and *LAPACKE_zgghd3*. It also exists with a native Fortran interface as *cgghd3*.

4.19.142 LAPACKE_cgghd3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgghd3_work(armpl_int_t matrix_layout, char compq,
                                char compz, armpl_int_t n, armpl_int_t ilo,
                                armpl_int_t ihi, armpl_singlecomplex_t *a,
```

(continues on next page)

(continued from previous page)

```
                                armpl_int_t lda, armpl_singlecomplex_t *b,
                                armpl_int_t ldb, armpl_singlecomplex_t *q,
                                armpl_int_t ldq, armpl_singlecomplex_t *z,
                                armpl_int_t ldz, armpl_singlecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgghd3_work*, *LAPACKE_dgghd3_work*, *LAPACKE_sgghd3_work* and *LAPACKE_zgghd3_work*.

4.19.143 LAPACKE_cgghrd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgghrd(armpl_int_t matrix_layout, char compq, char compz,
                           armpl_int_t n, armpl_int_t ilo, armpl_int_t ihi,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           armpl_singlecomplex_t *b, armpl_int_t ldb,
                           armpl_singlecomplex_t *q, armpl_int_t ldq,
                           armpl_singlecomplex_t *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgghrd*, *LAPACKE_dgghrd*, *LAPACKE_sgghrd* and *LAPACKE_zgghrd*. It also exists with a native Fortran interface as *cgghrd*.

4.19.144 LAPACKE_cgghrd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgghrd_work(armpl_int_t matrix_layout, char compq,
                                char compz, armpl_int_t n, armpl_int_t ilo,
                                armpl_int_t ihi, armpl_singlecomplex_t *a,
                                armpl_int_t lda, armpl_singlecomplex_t *b,
                                armpl_int_t ldb, armpl_singlecomplex_t *q,
                                armpl_int_t ldq, armpl_singlecomplex_t *z,
                                armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgghrd_work*, *LAPACKE_dgghrd_work*, *LAPACKE_sgghrd_work* and *LAPACKE_zgghrd_work*.

4.19.145 LAPACKE_cgglse

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgglse(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t p,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           armpl_singlecomplex_t *b, armpl_int_t ldb,
                           armpl_singlecomplex_t *c, armpl_singlecomplex_t *d,
                           armpl_singlecomplex_t *x);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgglse*, *LAPACKE_dgglse*, *LAPACKE_sgglse* and *LAPACKE_zgglse*. It also exists with a native Fortran interface as *cgglse*.

4.19.146 LAPACKE_cgglse_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgglse_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t p,
                                armpl_singlecomplex_t *a, armpl_int_t lda,
                                armpl_singlecomplex_t *b, armpl_int_t ldb,
                                armpl_singlecomplex_t *c,
                                armpl_singlecomplex_t *d,
                                armpl_singlecomplex_t *x,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgglse_work*, *LAPACKE_dgglse_work*, *LAPACKE_sgglse_work* and *LAPACKE_zgglse_work*.

4.19.147 LAPACKE_cggqrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cggqrf(armpl_int_t matrix_layout, armpl_int_t n,
                           armpl_int_t m, armpl_int_t p,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           armpl_singlecomplex_t *taua,
                           armpl_singlecomplex_t *b, armpl_int_t ldb,
                           armpl_singlecomplex_t *taub);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggqrf*, *LAPACKE_dggqrf*, *LAPACKE_sggqrf* and *LAPACKE_zggqrf*. It also exists with a native Fortran interface as *cggqrf*.

4.19.148 LAPACKE_cggqrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cggqrf_work(armpl_int_t matrix_layout, armpl_int_t n,
                                armpl_int_t m, armpl_int_t p,
                                armpl_singlecomplex_t *a, armpl_int_t lda,
                                armpl_singlecomplex_t *taua,
                                armpl_singlecomplex_t *b, armpl_int_t ldb,
                                armpl_singlecomplex_t *taub,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggqrf_work*, *LAPACKE_dggqrf_work*, *LAPACKE_sggqrf_work* and *LAPACKE_zggqrf_work*.

4.19.149 LAPACKE_cggrqf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cggrqf(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t p, armpl_int_t n,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           armpl_singlecomplex_t *taua,
                           armpl_singlecomplex_t *b, armpl_int_t ldb,
                           armpl_singlecomplex_t *taub);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggrqf*, *LAPACKE_dggrqf*, *LAPACKE_sggrqf* and *LAPACKE_zggrqf*. It also exists with a native Fortran interface as *cggrqf*.

4.19.150 LAPACKE_cggrqf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cggrqf_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t p, armpl_int_t n,
                                armpl_singlecomplex_t *a, armpl_int_t lda,
                                armpl_singlecomplex_t *taua,
                                armpl_singlecomplex_t *b, armpl_int_t ldb,
                                armpl_singlecomplex_t *taub,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggrqf_work*, *LAPACKE_dggrqf_work*, *LAPACKE_sggrqf_work* and *LAPACKE_zggrqf_work*.

4.19.151 LAPACKE_cggsvd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cggsvd(armpl_int_t matrix_layout, char jobu, char jobv,
                           char jobq, armpl_int_t m, armpl_int_t n,
                           armpl_int_t p, armpl_int_t *k, armpl_int_t *l,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           armpl_singlecomplex_t *b, armpl_int_t ldb,
                           float *alpha, float *beta,
                           armpl_singlecomplex_t *u, armpl_int_t ldu,
                           armpl_singlecomplex_t *v, armpl_int_t ldv,
                           armpl_singlecomplex_t *q, armpl_int_t ldq,
                           armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggsvd*, *LAPACKE_dggsvd*, *LAPACKE_sggsvd* and *LAPACKE_zggsvd*.

4.19.152 LAPACKE_cggsvd3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cggsvd3(armpl_int_t matrix_layout, char jobu, char jobv,
                            char jobq, armpl_int_t m, armpl_int_t n,
                            armpl_int_t p, armpl_int_t *k, armpl_int_t *l,
                            armpl_singlecomplex_t *a, armpl_int_t lda,
                            armpl_singlecomplex_t *b, armpl_int_t ldb,
                            float *alpha, float *beta,
                            armpl_singlecomplex_t *u, armpl_int_t ldu,
                            armpl_singlecomplex_t *v, armpl_int_t ldv,
                            armpl_singlecomplex_t *q, armpl_int_t ldq,
                            armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggsvd3*, *LAPACKE_dggsvd3*, *LAPACKE_sggsvd3* and *LAPACKE_zggsvd3*. It also exists with a native Fortran interface as *cggsvd3*.

4.19.153 LAPACKE_cggsvd3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cggsvd3_work(armpl_int_t matrix_layout, char jobu,
                                 char jobv, char jobq, armpl_int_t m,
                                 armpl_int_t n, armpl_int_t p, armpl_int_t *k,
                                 armpl_int_t *l, armpl_singlecomplex_t *a,
                                 armpl_int_t lda, armpl_singlecomplex_t *b,
                                 armpl_int_t ldb, float *alpha, float *beta,
                                 armpl_singlecomplex_t *u, armpl_int_t ldu,
                                 armpl_singlecomplex_t *v, armpl_int_t ldv,
                                 armpl_singlecomplex_t *q, armpl_int_t ldq,
                                 armpl_singlecomplex_t *work,
                                 armpl_int_t lwork, float *rwork,
                                 armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggsvd3_work*, *LAPACKE_dggsvd3_work*, *LAPACKE_sggsvd3_work* and *LAPACKE_zggsvd3_work*.

4.19.154 LAPACKE_cggsvd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cggsvd_work(armpl_int_t matrix_layout, char jobu,
                                char jobv, char jobq, armpl_int_t m,
                                armpl_int_t n, armpl_int_t p, armpl_int_t *k,
                                armpl_int_t *l, armpl_singlecomplex_t *a,
                                armpl_int_t lda, armpl_singlecomplex_t *b,
                                armpl_int_t ldb, float *alpha, float *beta,
                                armpl_singlecomplex_t *u, armpl_int_t ldu,
                                armpl_singlecomplex_t *v, armpl_int_t ldv,
                                armpl_singlecomplex_t *q, armpl_int_t ldq,
```

(continues on next page)

(continued from previous page)

```
                                armpl_singlecomplex_t *work, float *rwork,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggsvd_work*, *LAPACKE_dggsvd_work*, *LAPACKE_sggsvd_work* and *LAPACKE_zggsvd_work*.

4.19.155 LAPACKE_cggsvp

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cggsvp(armpl_int_t matrix_layout, char jobu, char jobv,
                           char jobq, armpl_int_t m, armpl_int_t p,
                           armpl_int_t n, armpl_singlecomplex_t *a,
                           armpl_int_t lda, armpl_singlecomplex_t *b,
                           armpl_int_t ldb, float tola, float tolb,
                           armpl_int_t *k, armpl_int_t *l,
                           armpl_singlecomplex_t *u, armpl_int_t ldu,
                           armpl_singlecomplex_t *v, armpl_int_t ldv,
                           armpl_singlecomplex_t *q, armpl_int_t ldq);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggsvp*, *LAPACKE_dggsvp*, *LAPACKE_sggsvp* and *LAPACKE_zggsvp*.

4.19.156 LAPACKE_cggsvp3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cggsvp3(armpl_int_t matrix_layout, char jobu, char jobv,
                            char jobq, armpl_int_t m, armpl_int_t p,
                            armpl_int_t n, armpl_singlecomplex_t *a,
                            armpl_int_t lda, armpl_singlecomplex_t *b,
                            armpl_int_t ldb, float tola, float tolb,
                            armpl_int_t *k, armpl_int_t *l,
                            armpl_singlecomplex_t *u, armpl_int_t ldu,
                            armpl_singlecomplex_t *v, armpl_int_t ldv,
                            armpl_singlecomplex_t *q, armpl_int_t ldq);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggsvp3*, *LAPACKE_dggsvp3*, *LAPACKE_sggsvp3* and *LAPACKE_zggsvp3*. It also exists with a native Fortran interface as *cggsvp3*.

4.19.157 LAPACKE_cggsvp3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cggsvp3_work(armpl_int_t matrix_layout, char jobu,
                                 char jobv, char jobq, armpl_int_t m,
                                 armpl_int_t p, armpl_int_t n,
                                 armpl_singlecomplex_t *a, armpl_int_t lda,
                                 armpl_singlecomplex_t *b, armpl_int_t ldb,
                                 float tola, float tolb, armpl_int_t *k,
                                 armpl_int_t *l, armpl_singlecomplex_t *u,
                                 armpl_int_t ldu, armpl_singlecomplex_t *v,
                                 armpl_int_t ldv, armpl_singlecomplex_t *q,
                                 armpl_int_t ldq, armpl_int_t *iwork,
                                 float *rwork, armpl_singlecomplex_t *tau,
                                 armpl_singlecomplex_t *work,
                                 armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggsvp3_work*, *LAPACKE_dggsvp3_work*, *LAPACKE_sggsvp3_work* and *LAPACKE_zggsvp3_work*.

4.19.158 LAPACKE_cggsvp_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cggsvp_work(armpl_int_t matrix_layout, char jobu,
                                char jobv, char jobq, armpl_int_t m,
                                armpl_int_t p, armpl_int_t n,
                                armpl_singlecomplex_t *a, armpl_int_t lda,
                                armpl_singlecomplex_t *b, armpl_int_t ldb,
                                float tola, float tolb, armpl_int_t *k,
                                armpl_int_t *l, armpl_singlecomplex_t *u,
                                armpl_int_t ldu, armpl_singlecomplex_t *v,
                                armpl_int_t ldv, armpl_singlecomplex_t *q,
                                armpl_int_t ldq, armpl_int_t *iwork,
                                float *rwork, armpl_singlecomplex_t *tau,
                                armpl_singlecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggsvp_work*, *LAPACKE_dggsvp_work*, *LAPACKE_sggsvp_work* and *LAPACKE_zggsvp_work*.

4.19.159 LAPACKE_cgtcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgtcon(char norm, armpl_int_t n,
                           const armpl_singlecomplex_t *dl,
                           const armpl_singlecomplex_t *d,
                           const armpl_singlecomplex_t *du,
                           const armpl_singlecomplex_t *du2,
                           const armpl_int_t *ipiv, float anorm,
                           float *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgtcon*, *LAPACKE_dgtcon*, *LAPACKE_sgtcon* and *LAPACKE_zgtcon*. It also exists with a native Fortran interface as *cgtcon*.

4.19.160 LAPACKE_cgtcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgtcon_work(char norm, armpl_int_t n,
                                const armpl_singlecomplex_t *dl,
                                const armpl_singlecomplex_t *d,
                                const armpl_singlecomplex_t *du,
                                const armpl_singlecomplex_t *du2,
                                const armpl_int_t *ipiv, float anorm,
                                float *rcond, armpl_singlecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgtcon_work*, *LAPACKE_dgtcon_work*, *LAPACKE_sgtcon_work* and *LAPACKE_zgtcon_work*.

4.19.161 LAPACKE_cgtrfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgtrfs(armpl_int_t matrix_layout, char trans,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_singlecomplex_t *dl,
                           const armpl_singlecomplex_t *d,
                           const armpl_singlecomplex_t *du,
                           const armpl_singlecomplex_t *dlf,
                           const armpl_singlecomplex_t *df,
                           const armpl_singlecomplex_t *duf,
                           const armpl_singlecomplex_t *du2,
                           const armpl_int_t *ipiv,
                           const armpl_singlecomplex_t *b, armpl_int_t ldb,
                           armpl_singlecomplex_t *x, armpl_int_t ldx,
                           float *ferr, float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgtrfs*, *LAPACKE_dgtrfs*, *LAPACKE_sgtrfs* and *LAPACKE_zgtrfs*. It also exists with a native Fortran interface as *cgtrfs*.

4.19.162 LAPACKE_cgtrfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgtrfs_work(armpl_int_t matrix_layout, char trans,
                                armpl_int_t n, armpl_int_t nrhs,
                                const armpl_singlecomplex_t *dl,
                                const armpl_singlecomplex_t *d,
                                const armpl_singlecomplex_t *du,
                                const armpl_singlecomplex_t *dlf,
                                const armpl_singlecomplex_t *df,
                                const armpl_singlecomplex_t *duf,
                                const armpl_singlecomplex_t *du2,
                                const armpl_int_t *ipiv,
                                const armpl_singlecomplex_t *b,
                                armpl_int_t ldb, armpl_singlecomplex_t *x,
                                armpl_int_t ldx, float *ferr, float *berr,
                                armpl_singlecomplex_t *work, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgtrfs_work*, *LAPACKE_dgtrfs_work*, *LAPACKE_sgtrfs_work* and *LAPACKE_zgtrfs_work*.

4.19.163 LAPACKE_cgtsv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgtsv(armpl_int_t matrix_layout, armpl_int_t n,
                          armpl_int_t nrhs, armpl_singlecomplex_t *dl,
                          armpl_singlecomplex_t *d, armpl_singlecomplex_t *du,
                          armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgtsv*, *LAPACKE_dgtsv*, *LAPACKE_sgtsv* and *LAPACKE_zgtsv*. It also exists with a native Fortran interface as *cgtsv*.

4.19.164 LAPACKE_cgtsv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgtsv_work(armpl_int_t matrix_layout, armpl_int_t n,
                               armpl_int_t nrhs, armpl_singlecomplex_t *dl,
                               armpl_singlecomplex_t *d,
                               armpl_singlecomplex_t *du,
                               armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgtsv_work*, *LAPACKE_dgtsv_work*, *LAPACKE_sgtsv_work* and *LAPACKE_zgtsv_work*.

4.19.165 LAPACKE_cgtsvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgtsvx(armpl_int_t matrix_layout, char fact, char trans,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_singlecomplex_t *dl,
                           const armpl_singlecomplex_t *d,
                           const armpl_singlecomplex_t *du,
                           armpl_singlecomplex_t *dlf,
                           armpl_singlecomplex_t *df,
                           armpl_singlecomplex_t *duf,
                           armpl_singlecomplex_t *du2, armpl_int_t *ipiv,
                           const armpl_singlecomplex_t *b, armpl_int_t ldb,
                           armpl_singlecomplex_t *x, armpl_int_t ldx,
                           float *rcond, float *ferr, float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgtsvx*, *LAPACKE_dgtsvx*, *LAPACKE_sgtsvx* and *LAPACKE_zgtsvx*. It also exists with a native Fortran interface as *cgtsvx*.

4.19.166 LAPACKE_cgtsvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgtsvx_work(armpl_int_t matrix_layout, char fact,
                                char trans, armpl_int_t n, armpl_int_t nrhs,
                                const armpl_singlecomplex_t *dl,
                                const armpl_singlecomplex_t *d,
                                const armpl_singlecomplex_t *du,
                                armpl_singlecomplex_t *dlf,
                                armpl_singlecomplex_t *df,
                                armpl_singlecomplex_t *duf,
                                armpl_singlecomplex_t *du2, armpl_int_t *ipiv,
                                const armpl_singlecomplex_t *b,
                                armpl_int_t ldb, armpl_singlecomplex_t *x,
                                armpl_int_t ldx, float *rcond, float *ferr,
                                float *berr, armpl_singlecomplex_t *work,
                                float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgtsvx_work*, *LAPACKE_dgtsvx_work*, *LAPACKE_sgtsvx_work* and *LAPACKE_zgtsvx_work*.

4.19.167 LAPACKE_cgttrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgttrf(armpl_int_t n, armpl_singlecomplex_t *dl,
                           armpl_singlecomplex_t *d,
                           armpl_singlecomplex_t *du,
                           armpl_singlecomplex_t *du2, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgttrf*, *LAPACKE_dgttrf*, *LAPACKE_sgttrf* and *LAPACKE_zgttrf*. It also exists with a native Fortran interface as *cgttrf*.

4.19.168 LAPACKE_cgttrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgttrf_work(armpl_int_t n, armpl_singlecomplex_t *dl,
                                armpl_singlecomplex_t *d,
                                armpl_singlecomplex_t *du,
                                armpl_singlecomplex_t *du2,
                                armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgttrf_work*, *LAPACKE_dgttrf_work*, *LAPACKE_sgttrf_work* and *LAPACKE_zgttrf_work*.

4.19.169 LAPACKE_cgttrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgttrs(armpl_int_t matrix_layout, char trans,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_singlecomplex_t *dl,
                           const armpl_singlecomplex_t *d,
                           const armpl_singlecomplex_t *du,
                           const armpl_singlecomplex_t *du2,
                           const armpl_int_t *ipiv, armpl_singlecomplex_t *b,
                           armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgttrs*, *LAPACKE_dgttrs*, *LAPACKE_sgttrs* and *LAPACKE_zgttrs*. It also exists with a native Fortran interface as *cgttrs*.

4.19.170 LAPACKE_cgttrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cgttrs_work(armpl_int_t matrix_layout, char trans,
                                armpl_int_t n, armpl_int_t nrhs,
                                const armpl_singlecomplex_t *dl,
                                const armpl_singlecomplex_t *d,
                                const armpl_singlecomplex_t *du,
                                const armpl_singlecomplex_t *du2,
                                const armpl_int_t *ipiv,
                                armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgttrs_work*, *LAPACKE_dgttrs_work*, *LAPACKE_sgttrs_work* and *LAPACKE_zgttrs_work*.

4.19.171 LAPACKE_chbev

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chbev(armpl_int_t matrix_layout, char jobz, char uplo,
                          armpl_int_t n, armpl_int_t kd,
                          armpl_singlecomplex_t *ab, armpl_int_t ldab,
                          float *w, armpl_singlecomplex_t *z,
                          armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.172 LAPACKE_chbev_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chbev_2stage(armpl_int_t matrix_layout, char jobz,
                                 char uplo, armpl_int_t n, armpl_int_t kd,
                                 armpl_singlecomplex_t *ab, armpl_int_t ldab,
                                 float *w, armpl_singlecomplex_t *z,
                                 armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbev_2stage* and *LAPACKE_zhbev_2stage*. It also exists with a native Fortran interface as *chbev_2stage*.

4.19.173 LAPACKE_chbev_2stage_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chbev_2stage_work(armpl_int_t matrix_layout, char jobz,
                                      char uplo, armpl_int_t n,
                                      armpl_int_t kd,
                                      armpl_singlecomplex_t *ab,
                                      armpl_int_t ldab, float *w,
                                      armpl_singlecomplex_t *z,
                                      armpl_int_t ldz,
                                      armpl_singlecomplex_t *work,
                                      armpl_int_t lwork, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbev_2stage_work* and *LAPACKE_zhbev_2stage_work*.

4.19.174 LAPACKE_chbev_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chbev_work(armpl_int_t matrix_layout, char jobz,
                               char uplo, armpl_int_t n, armpl_int_t kd,
                               armpl_singlecomplex_t *ab, armpl_int_t ldab,
                               float *w, armpl_singlecomplex_t *z,
                               armpl_int_t ldz, armpl_singlecomplex_t *work,
                               float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbev_work* and *LAPACKE_zhbev_work*.

4.19.175 LAPACKE_chbevd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chbevd(armpl_int_t matrix_layout, char jobz, char uplo,
                           armpl_int_t n, armpl_int_t kd,
                           armpl_singlecomplex_t *ab, armpl_int_t ldab,
                           float *w, armpl_singlecomplex_t *z,
                           armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbevd* and *LAPACKE_zhbevd*. It also exists with a native Fortran interface as *chbevd*.

4.19.176 LAPACKE_chbevd_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chbevd_2stage(armpl_int_t matrix_layout, char jobz,
                                  char uplo, armpl_int_t n, armpl_int_t kd,
                                  armpl_singlecomplex_t *ab, armpl_int_t ldab,
                                  float *w, armpl_singlecomplex_t *z,
                                  armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbevd_2stage* and *LAPACKE_zhbevd_2stage*. It also exists with a native Fortran interface as *chbevd_2stage*.

4.19.177 LAPACKE_chbevd_2stage_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chbevd_2stage_work(armpl_int_t matrix_layout, char jobz,
                                       char uplo, armpl_int_t n,
                                       armpl_int_t kd,
                                       armpl_singlecomplex_t *ab,
```

(continues on next page)

(continued from previous page)

```
                                        armpl_int_t ldab, float *w,
                                        armpl_singlecomplex_t *z,
                                        armpl_int_t ldz,
                                        armpl_singlecomplex_t *work,
                                        armpl_int_t lwork, float *rwork,
                                        armpl_int_t lrwork, armpl_int_t *iwork,
                                        armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbevd_2stage_work* and *LAPACKE_zhbevd_2stage_work*.

4.19.178 LAPACKE_chbevd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chbevd_work(armpl_int_t matrix_layout, char jobz,
                                char uplo, armpl_int_t n, armpl_int_t kd,
                                armpl_singlecomplex_t *ab, armpl_int_t ldab,
                                float *w, armpl_singlecomplex_t *z,
                                armpl_int_t ldz, armpl_singlecomplex_t *work,
                                armpl_int_t lwork, float *rwork,
                                armpl_int_t lrwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbevd_work* and *LAPACKE_zhbevd_work*.

4.19.179 LAPACKE_chbevx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chbevx(armpl_int_t matrix_layout, char jobz, char range,
                           char uplo, armpl_int_t n, armpl_int_t kd,
                           armpl_singlecomplex_t *ab, armpl_int_t ldab,
                           armpl_singlecomplex_t *q, armpl_int_t ldq,
                           float vl, float vu, armpl_int_t il, armpl_int_t iu,
                           float abstol, armpl_int_t *m, float *w,
                           armpl_singlecomplex_t *z, armpl_int_t ldz,
                           armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbevx* and *LAPACKE_zhbevx*. It also exists with a native Fortran interface as *chbevx*.

4.19.180 LAPACKE_chbevx_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chbevx_2stage(armpl_int_t matrix_layout, char jobz,
                                  char range, char uplo, armpl_int_t n,
                                  armpl_int_t kd, armpl_singlecomplex_t *ab,
                                  armpl_int_t ldab, armpl_singlecomplex_t *q,
                                  armpl_int_t ldq, float vl, float vu,
                                  armpl_int_t il, armpl_int_t iu,
                                  float abstol, armpl_int_t *m, float *w,
                                  armpl_singlecomplex_t *z, armpl_int_t ldz,
                                  armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbevx_2stage* and *LAPACKE_zhbevx_2stage*. It also exists with a native Fortran interface as *chbevx_2stage*.

4.19.181 LAPACKE_chbevx_2stage_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chbevx_2stage_work(armpl_int_t matrix_layout, char jobz,
                                       char range, char uplo, armpl_int_t n,
                                       armpl_int_t kd,
                                       armpl_singlecomplex_t *ab,
                                       armpl_int_t ldab,
                                       armpl_singlecomplex_t *q,
                                       armpl_int_t ldq, float vl, float vu,
                                       armpl_int_t il, armpl_int_t iu,
                                       float abstol, armpl_int_t *m, float *w,
                                       armpl_singlecomplex_t *z,
                                       armpl_int_t ldz,
                                       armpl_singlecomplex_t *work,
                                       armpl_int_t lwork, float *rwork,
                                       armpl_int_t *iwork,
                                       armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbevx_2stage_work* and *LAPACKE_zhbevx_2stage_work*.

4.19.182 LAPACKE_chbevx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chbevx_work(armpl_int_t matrix_layout, char jobz,
                                char range, char uplo, armpl_int_t n,
                                armpl_int_t kd, armpl_singlecomplex_t *ab,
                                armpl_int_t ldab, armpl_singlecomplex_t *q,
                                armpl_int_t ldq, float vl, float vu,
                                armpl_int_t il, armpl_int_t iu, float abstol,
                                armpl_int_t *m, float *w,
                                armpl_singlecomplex_t *z, armpl_int_t ldz,
                                armpl_singlecomplex_t *work, float *rwork,
                                armpl_int_t *iwork, armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbevx_work* and *LAPACKE_zhbevx_work*.

4.19.183 LAPACKE_chbgst

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chbgst(armpl_int_t matrix_layout, char vect, char uplo,
                           armpl_int_t n, armpl_int_t ka, armpl_int_t kb,
                           armpl_singlecomplex_t *ab, armpl_int_t ldab,
                           const armpl_singlecomplex_t *bb, armpl_int_t ldbb,
                           armpl_singlecomplex_t *x, armpl_int_t ldx);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbgst* and *LAPACKE_zhbgst*. It also exists with a native Fortran interface as *chbgst*.

4.19.184 LAPACKE_chbgst_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chbgst_work(armpl_int_t matrix_layout, char vect,
                                char uplo, armpl_int_t n, armpl_int_t ka,
                                armpl_int_t kb, armpl_singlecomplex_t *ab,
                                armpl_int_t ldab,
                                const armpl_singlecomplex_t *bb,
                                armpl_int_t ldbb, armpl_singlecomplex_t *x,
                                armpl_int_t ldx, armpl_singlecomplex_t *work,
                                float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbgst_work* and *LAPACKE_zhbgst_work*.

4.19.185 LAPACKE_chbgv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chbgv(armpl_int_t matrix_layout, char jobz, char uplo,
                          armpl_int_t n, armpl_int_t ka, armpl_int_t kb,
                          armpl_singlecomplex_t *ab, armpl_int_t ldab,
                          armpl_singlecomplex_t *bb, armpl_int_t ldbb,
                          float *w, armpl_singlecomplex_t *z,
                          armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbgv* and *LAPACKE_zhbgv*. It also exists with a native Fortran interface as *chbgv*.

4.19.186 LAPACKE_chbgv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chbgv_work(armpl_int_t matrix_layout, char jobz,
                               char uplo, armpl_int_t n, armpl_int_t ka,
                               armpl_int_t kb, armpl_singlecomplex_t *ab,
                               armpl_int_t ldab, armpl_singlecomplex_t *bb,
                               armpl_int_t ldbb, float *w,
                               armpl_singlecomplex_t *z, armpl_int_t ldz,
                               armpl_singlecomplex_t *work, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbgv_work* and *LAPACKE_zhbgv_work*.

4.19.187 LAPACKE_chbgvd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chbgvd(armpl_int_t matrix_layout, char jobz, char uplo,
                           armpl_int_t n, armpl_int_t ka, armpl_int_t kb,
                           armpl_singlecomplex_t *ab, armpl_int_t ldab,
                           armpl_singlecomplex_t *bb, armpl_int_t ldbb,
                           float *w, armpl_singlecomplex_t *z,
                           armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbgvd* and *LAPACKE_zhbgvd*. It also exists with a native Fortran interface as *chbgvd*.

4.19.188 LAPACKE_chbgvd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chbgvd_work(armpl_int_t matrix_layout, char jobz,
                                char uplo, armpl_int_t n, armpl_int_t ka,
                                armpl_int_t kb, armpl_singlecomplex_t *ab,
                                armpl_int_t ldab, armpl_singlecomplex_t *bb,
                                armpl_int_t ldbb, float *w,
                                armpl_singlecomplex_t *z, armpl_int_t ldz,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork, float *rwork,
                                armpl_int_t lrwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbgvd_work* and *LAPACKE_zhbgvd_work*.

4.19.189 LAPACKE_chbgvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chbgvx(armpl_int_t matrix_layout, char jobz, char range,
                           char uplo, armpl_int_t n, armpl_int_t ka,
                           armpl_int_t kb, armpl_singlecomplex_t *ab,
                           armpl_int_t ldab, armpl_singlecomplex_t *bb,
                           armpl_int_t ldbb, armpl_singlecomplex_t *q,
                           armpl_int_t ldq, float vl, float vu,
                           armpl_int_t il, armpl_int_t iu, float abstol,
```

(continues on next page)

(continued from previous page)

```
                                armpl_int_t *m, float *w, armpl_singlecomplex_t *z,
                                armpl_int_t ldz, armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbgvx* and *LAPACKE_zhbgvx*. It also exists with a native Fortran interface as *chbgvx*.

4.19.190 LAPACKE_chbgvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chbgvx_work(armpl_int_t matrix_layout, char jobz,
                                char range, char uplo, armpl_int_t n,
                                armpl_int_t ka, armpl_int_t kb,
                                armpl_singlecomplex_t *ab, armpl_int_t ldab,
                                armpl_singlecomplex_t *bb, armpl_int_t ldbb,
                                armpl_singlecomplex_t *q, armpl_int_t ldq,
                                float vl, float vu, armpl_int_t il,
                                armpl_int_t iu, float abstol, armpl_int_t *m,
                                float *w, armpl_singlecomplex_t *z,
                                armpl_int_t ldz, armpl_singlecomplex_t *work,
                                float *rwork, armpl_int_t *iwork,
                                armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbgvx_work* and *LAPACKE_zhbgvx_work*.

4.19.191 LAPACKE_chbtrd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chbtrd(armpl_int_t matrix_layout, char vect, char uplo,
                           armpl_int_t n, armpl_int_t kd,
                           armpl_singlecomplex_t *ab, armpl_int_t ldab,
                           float *d, float *e, armpl_singlecomplex_t *q,
                           armpl_int_t ldq);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbtrd* and *LAPACKE_zhbtrd*. It also exists with a native Fortran interface as *chbtrd*.

4.19.192 LAPACKE_chbtrd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chbtrd_work(armpl_int_t matrix_layout, char vect,
                                char uplo, armpl_int_t n, armpl_int_t kd,
                                armpl_singlecomplex_t *ab, armpl_int_t ldab,
                                float *d, float *e, armpl_singlecomplex_t *q,
                                armpl_int_t ldq,
                                armpl_singlecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbtrd_work* and *LAPACKE_zhbtrd_work*.

4.19.193 LAPACKE_checon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_checon(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, const armpl_singlecomplex_t *a,
                           armpl_int_t lda, const armpl_int_t *ipiv,
                           float anorm, float *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_checon* and *LAPACKE_zhecon*. It also exists with a native Fortran interface as *checon*.

4.19.194 LAPACKE_checon_3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_checon_3(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, const armpl_singlecomplex_t *a,
                             armpl_int_t lda, const armpl_singlecomplex_t *e,
                             const armpl_int_t *ipiv, float anorm,
                             float *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_checon_3* and *LAPACKE_zhecon_3*. It also exists with a native Fortran interface as *checon_3*.

4.19.195 LAPACKE_checon_3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_checon_3_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n,
                                  const armpl_singlecomplex_t *a,
                                  armpl_int_t lda,
                                  const armpl_singlecomplex_t *e,
                                  const armpl_int_t *ipiv, float anorm,
                                  float *rcond, armpl_singlecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_checon_3_work* and *LAPACKE_zhecon_3_work*.

4.19.196 LAPACKE_checon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_checon_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, const armpl_singlecomplex_t *a,
                                armpl_int_t lda, const armpl_int_t *ipiv,
                                float anorm, float *rcond,
                                armpl_singlecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_checon_work* and *LAPACKE_zhecon_work*.

4.19.197 LAPACKE_cheequb

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cheequb(armpl_int_t matrix_layout, char uplo,
                            armpl_int_t n, const armpl_singlecomplex_t *a,
                            armpl_int_t lda, float *s, float *scond,
                            float *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheequb* and *LAPACKE_zheequb*. It also exists with a native Fortran interface as *cheequb*.

4.19.198 LAPACKE_cheequb_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cheequb_work(armpl_int_t matrix_layout, char uplo,
                                 armpl_int_t n,
                                 const armpl_singlecomplex_t *a,
                                 armpl_int_t lda, float *s, float *scond,
                                 float *amax, armpl_singlecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheequb_work* and *LAPACKE_zheequb_work*.

4.19.199 LAPACKE_cheev

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cheev(armpl_int_t matrix_layout, char jobz, char uplo,
                          armpl_int_t n, armpl_singlecomplex_t *a,
                          armpl_int_t lda, float *w);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheev* and *LAPACKE_zheev*. It also exists with a native Fortran interface as *cheev*.

4.19.200 LAPACKE_cheev_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cheev_2stage(armpl_int_t matrix_layout, char jobz,
                                 char uplo, armpl_int_t n,
                                 armpl_singlecomplex_t *a, armpl_int_t lda,
                                 float *w);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheev_2stage* and *LAPACKE_zheev_2stage*. It also exists with a native Fortran interface as *cheev_2stage*.

4.19.201 LAPACKE_cheev_2stage_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cheev_2stage_work(armpl_int_t matrix_layout, char jobz,
                                      char uplo, armpl_int_t n,
                                      armpl_singlecomplex_t *a,
                                      armpl_int_t lda, float *w,
                                      armpl_singlecomplex_t *work,
                                      armpl_int_t lwork, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheev_2stage_work* and *LAPACKE_zheev_2stage_work*.

4.19.202 LAPACKE_cheev_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cheev_work(armpl_int_t matrix_layout, char jobz,
                               char uplo, armpl_int_t n,
                               armpl_singlecomplex_t *a, armpl_int_t lda,
                               float *w, armpl_singlecomplex_t *work,
                               armpl_int_t lwork, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheev_work* and *LAPACKE_zheev_work*.

4.19.203 LAPACKE_cheevd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cheevd(armpl_int_t matrix_layout, char jobz, char uplo,
                           armpl_int_t n, armpl_singlecomplex_t *a,
                           armpl_int_t lda, float *w);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheevd* and *LAPACKE_zheevd*. It also exists with a native Fortran interface as *cheevd*.

4.19.204 LAPACKE_cheevd_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cheevd_2stage(armpl_int_t matrix_layout, char jobz,
                                  char uplo, armpl_int_t n,
                                  armpl_singlecomplex_t *a, armpl_int_t lda,
                                  float *w);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheevd_2stage* and *LAPACKE_zheevd_2stage*. It also exists with a native Fortran interface as *cheevd_2stage*.

4.19.205 LAPACKE_cheevd_2stage_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cheevd_2stage_work(armpl_int_t matrix_layout, char jobz,
                                       char uplo, armpl_int_t n,
                                       armpl_singlecomplex_t *a,
                                       armpl_int_t lda, float *w,
                                       armpl_singlecomplex_t *work,
                                       armpl_int_t lwork, float *rwork,
                                       armpl_int_t lrwork, armpl_int_t *iwork,
                                       armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheevd_2stage_work* and *LAPACKE_zheevd_2stage_work*.

4.19.206 LAPACKE_cheevd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cheevd_work(armpl_int_t matrix_layout, char jobz,
                                char uplo, armpl_int_t n,
                                armpl_singlecomplex_t *a, armpl_int_t lda,
                                float *w, armpl_singlecomplex_t *work,
                                armpl_int_t lwork, float *rwork,
                                armpl_int_t lrwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheevd_work* and *LAPACKE_zheevd_work*.

4.19.207 LAPACKE_cheevr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cheevr(armpl_int_t matrix_layout, char jobz, char range,
                           char uplo, armpl_int_t n, armpl_singlecomplex_t *a,
                           armpl_int_t lda, float vl, float vu,
                           armpl_int_t il, armpl_int_t iu, float abstol,
                           armpl_int_t *m, float *w, armpl_singlecomplex_t *z,
                           armpl_int_t ldz, armpl_int_t *isuppz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheevr* and *LAPACKE_zheevr*. It also exists with a native Fortran interface as *cheevr*.

4.19.208 LAPACKE_cheevr_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cheevr_2stage(armpl_int_t matrix_layout, char jobz,
                                  char range, char uplo, armpl_int_t n,
                                  armpl_singlecomplex_t *a, armpl_int_t lda,
                                  float vl, float vu, armpl_int_t il,
                                  armpl_int_t iu, float abstol,
                                  armpl_int_t *m, float *w,
                                  armpl_singlecomplex_t *z, armpl_int_t ldz,
                                  armpl_int_t *isuppz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheevr_2stage* and *LAPACKE_zheevr_2stage*. It also exists with a native Fortran interface as *cheevr_2stage*.

4.19.209 LAPACKE_cheevr_2stage_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cheevr_2stage_work(armpl_int_t matrix_layout, char jobz,
                                       char range, char uplo, armpl_int_t n,
                                       armpl_singlecomplex_t *a,
                                       armpl_int_t lda, float vl, float vu,
                                       armpl_int_t il, armpl_int_t iu,
                                       float abstol, armpl_int_t *m, float *w,
                                       armpl_singlecomplex_t *z,
                                       armpl_int_t ldz, armpl_int_t *isuppz,
                                       armpl_singlecomplex_t *work,
                                       armpl_int_t lwork, float *rwork,
                                       armpl_int_t lrwork, armpl_int_t *iwork,
                                       armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheevr_2stage_work* and *LAPACKE_zheevr_2stage_work*.

4.19.210 LAPACKE_cheevr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cheevr_work(armpl_int_t matrix_layout, char jobz,
                                char range, char uplo, armpl_int_t n,
                                armpl_singlecomplex_t *a, armpl_int_t lda,
                                float vl, float vu, armpl_int_t il,
                                armpl_int_t iu, float abstol, armpl_int_t *m,
```

(continues on next page)

(continued from previous page)

```
                                    float *w, armpl_singlecomplex_t *z,
                                    armpl_int_t ldz, armpl_int_t *isuppz,
                                    armpl_singlecomplex_t *work,
                                    armpl_int_t lwork, float *rwork,
                                    armpl_int_t lrwork, armpl_int_t *iwork,
                                    armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheevr_work* and *LAPACKE_zheevr_work*.

4.19.211 LAPACKE_cheevx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cheevx(armpl_int_t matrix_layout, char jobz, char range,
                           char uplo, armpl_int_t n, armpl_singlecomplex_t *a,
                           armpl_int_t lda, float vl, float vu,
                           armpl_int_t il, armpl_int_t iu, float abstol,
                           armpl_int_t *m, float *w, armpl_singlecomplex_t *z,
                           armpl_int_t ldz, armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheevx* and *LAPACKE_zheevx*. It also exists with a native Fortran interface as *cheevx*.

4.19.212 LAPACKE_cheevx_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cheevx_2stage(armpl_int_t matrix_layout, char jobz,
                                  char range, char uplo, armpl_int_t n,
                                  armpl_singlecomplex_t *a, armpl_int_t lda,
                                  float vl, float vu, armpl_int_t il,
                                  armpl_int_t iu, float abstol,
                                  armpl_int_t *m, float *w,
                                  armpl_singlecomplex_t *z, armpl_int_t ldz,
                                  armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheevx_2stage* and *LAPACKE_zheevx_2stage*. It also exists with a native Fortran interface as *cheevx_2stage*.

4.19.213 LAPACKE_cheevx_2stage_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cheevx_2stage_work(armpl_int_t matrix_layout, char jobz,
                                       char range, char uplo, armpl_int_t n,
                                       armpl_singlecomplex_t *a,
                                       armpl_int_t lda, float vl, float vu,
                                       armpl_int_t il, armpl_int_t iu,
                                       float abstol, armpl_int_t *m, float *w,
                                       armpl_singlecomplex_t *z,
                                       armpl_int_t ldz,
                                       armpl_singlecomplex_t *work,
                                       armpl_int_t lwork, float *rwork,
                                       armpl_int_t *iwork,
                                       armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheevx_2stage_work* and *LAPACKE_zheevx_2stage_work*.

4.19.214 LAPACKE_cheevx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cheevx_work(armpl_int_t matrix_layout, char jobz,
                                char range, char uplo, armpl_int_t n,
                                armpl_singlecomplex_t *a, armpl_int_t lda,
                                float vl, float vu, armpl_int_t il,
                                armpl_int_t iu, float abstol, armpl_int_t *m,
                                float *w, armpl_singlecomplex_t *z,
                                armpl_int_t ldz, armpl_singlecomplex_t *work,
                                armpl_int_t lwork, float *rwork,
                                armpl_int_t *iwork, armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheevx_work* and *LAPACKE_zheevx_work*.

4.19.215 LAPACKE_chegst

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chegst(armpl_int_t matrix_layout, armpl_int_t itype,
                           char uplo, armpl_int_t n, armpl_singlecomplex_t *a,
                           armpl_int_t lda, const armpl_singlecomplex_t *b,
                           armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chegst* and *LAPACKE_zhegst*. It also exists with a native Fortran interface as *chegst*.

4.19.216 LAPACKE_chegst_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chegst_work(armpl_int_t matrix_layout, armpl_int_t itype,
                                char uplo, armpl_int_t n,
                                armpl_singlecomplex_t *a, armpl_int_t lda,
                                const armpl_singlecomplex_t *b,
                                armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chegst_work* and *LAPACKE_zhegst_work*.

4.19.217 LAPACKE_chegv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chegv(armpl_int_t matrix_layout, armpl_int_t itype,
                          char jobz, char uplo, armpl_int_t n,
                          armpl_singlecomplex_t *a, armpl_int_t lda,
                          armpl_singlecomplex_t *b, armpl_int_t ldb,
                          float *w);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chegv* and *LAPACKE_zhegv*. It also exists with a native Fortran interface as *chegv*.

4.19.218 LAPACKE_chegv_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chegv_2stage(armpl_int_t matrix_layout, armpl_int_t itype,
                                 char jobz, char uplo, armpl_int_t n,
                                 armpl_singlecomplex_t *a, armpl_int_t lda,
                                 armpl_singlecomplex_t *b, armpl_int_t ldb,
                                 float *w);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chegv_2stage* and *LAPACKE_zhegv_2stage*. It also exists with a native Fortran interface as *chegv_2stage*.

4.19.219 LAPACKE_chegv_2stage_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chegv_2stage_work(armpl_int_t matrix_layout,
                                      armpl_int_t itype, char jobz, char uplo,
                                      armpl_int_t n, armpl_singlecomplex_t *a,
                                      armpl_int_t lda,
```

(continues on next page)

(continued from previous page)

```
                                    armpl_singlecomplex_t *b,
                                    armpl_int_t ldb, float *w,
                                    armpl_singlecomplex_t *work,
                                    armpl_int_t lwork, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chegv_2stage_work* and *LAPACKE_zhegv_2stage_work*.

4.19.220 LAPACKE_chegv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chegv_work(armpl_int_t matrix_layout, armpl_int_t itype,
                               char jobz, char uplo, armpl_int_t n,
                               armpl_singlecomplex_t *a, armpl_int_t lda,
                               armpl_singlecomplex_t *b, armpl_int_t ldb,
                               float *w, armpl_singlecomplex_t *work,
                               armpl_int_t lwork, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chegv_work* and *LAPACKE_zhegv_work*.

4.19.221 LAPACKE_chegvd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chegvd(armpl_int_t matrix_layout, armpl_int_t itype,
                           char jobz, char uplo, armpl_int_t n,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           armpl_singlecomplex_t *b, armpl_int_t ldb,
                           float *w);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chegvd* and *LAPACKE_zhegvd*. It also exists with a native Fortran interface as *chegvd*.

4.19.222 LAPACKE_chegvd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chegvd_work(armpl_int_t matrix_layout, armpl_int_t itype,
                                char jobz, char uplo, armpl_int_t n,
                                armpl_singlecomplex_t *a, armpl_int_t lda,
                                armpl_singlecomplex_t *b, armpl_int_t ldb,
                                float *w, armpl_singlecomplex_t *work,
                                armpl_int_t lwork, float *rwork,
                                armpl_int_t lrwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chegvd_work* and *LAPACKE_zhegvd_work*.

4.19.223 LAPACKE_chegvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chegvx(armpl_int_t matrix_layout, armpl_int_t itype,
                           char jobz, char range, char uplo, armpl_int_t n,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           armpl_singlecomplex_t *b, armpl_int_t ldb,
                           float vl, float vu, armpl_int_t il, armpl_int_t iu,
                           float abstol, armpl_int_t *m, float *w,
                           armpl_singlecomplex_t *z, armpl_int_t ldz,
                           armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chegvx* and *LAPACKE_zhegvx*. It also exists with a native Fortran interface as *chegvx*.

4.19.224 LAPACKE_chegvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chegvx_work(armpl_int_t matrix_layout, armpl_int_t itype,
                                char jobz, char range, char uplo,
                                armpl_int_t n, armpl_singlecomplex_t *a,
                                armpl_int_t lda, armpl_singlecomplex_t *b,
                                armpl_int_t ldb, float vl, float vu,
                                armpl_int_t il, armpl_int_t iu, float abstol,
                                armpl_int_t *m, float *w,
                                armpl_singlecomplex_t *z, armpl_int_t ldz,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork, float *rwork,
                                armpl_int_t *iwork, armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chegvx_work* and *LAPACKE_zhegvx_work*.

4.19.225 LAPACKE_cherfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cherfs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_singlecomplex_t *a, armpl_int_t lda,
                           const armpl_singlecomplex_t *af, armpl_int_t ldaf,
                           const armpl_int_t *ipiv,
                           const armpl_singlecomplex_t *b, armpl_int_t ldb,
                           armpl_singlecomplex_t *x, armpl_int_t ldx,
                           float *ferr, float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cherfs* and *LAPACKE_zherfs*. It also exists with a native Fortran interface as *cherfs*.

4.19.226 LAPACKE_cherfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cherfs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
```

(continues on next page)

(continued from previous page)

```
                                const armpl_singlecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_singlecomplex_t *af,
                                armpl_int_t ldaf, const armpl_int_t *ipiv,
                                const armpl_singlecomplex_t *b,
                                armpl_int_t ldb, armpl_singlecomplex_t *x,
                                armpl_int_t ldx, float *ferr, float *berr,
                                armpl_singlecomplex_t *work, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cherfs_work* and *LAPACKE_zherfs_work*.

4.19.227 LAPACKE_cherfsx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cherfsx(armpl_int_t matrix_layout, char uplo, char equed,
                            armpl_int_t n, armpl_int_t nrhs,
                            const armpl_singlecomplex_t *a, armpl_int_t lda,
                            const armpl_singlecomplex_t *af, armpl_int_t ldaf,
                            const armpl_int_t *ipiv, const float *s,
                            const armpl_singlecomplex_t *b, armpl_int_t ldb,
                            armpl_singlecomplex_t *x, armpl_int_t ldx,
                            float *rcond, float *berr, armpl_int_t n_err_bnds,
                            float *err_bnds_norm, float *err_bnds_comp,
                            armpl_int_t nparams, float *params);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cherfsx* and *LAPACKE_zherfsx*. It also exists with a native Fortran interface as *cherfsx*.

4.19.228 LAPACKE_cherfsx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cherfsx_work(armpl_int_t matrix_layout, char uplo,
                                 char equed, armpl_int_t n, armpl_int_t nrhs,
                                 const armpl_singlecomplex_t *a,
                                 armpl_int_t lda,
                                 const armpl_singlecomplex_t *af,
                                 armpl_int_t ldaf, const armpl_int_t *ipiv,
                                 const float *s,
                                 const armpl_singlecomplex_t *b,
                                 armpl_int_t ldb, armpl_singlecomplex_t *x,
                                 armpl_int_t ldx, float *rcond, float *berr,
                                 armpl_int_t n_err_bnds, float *err_bnds_norm,
                                 float *err_bnds_comp, armpl_int_t nparams,
                                 float *params, armpl_singlecomplex_t *work,
                                 float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cherfsx_work* and *LAPACKE_zherfsx_work*.

4.19.229 LAPACKE_chesv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chesv(armpl_int_t matrix_layout, char uplo, armpl_int_t n,
                          armpl_int_t nrhs, armpl_singlecomplex_t *a,
                          armpl_int_t lda, armpl_int_t *ipiv,
                          armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chesv* and *LAPACKE_zhesv*. It also exists with a native Fortran interface as *chesv*.

4.19.230 LAPACKE_chesv_aa

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chesv_aa(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, armpl_int_t nrhs,
                             armpl_singlecomplex_t *a, armpl_int_t lda,
                             armpl_int_t *ipiv, armpl_singlecomplex_t *b,
                             armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chesv_aa* and *LAPACKE_zhesv_aa*. It also exists with a native Fortran interface as *chesv_aa*.

4.19.231 LAPACKE_chesv_aa_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chesv_aa_2stage(armpl_int_t matrix_layout, char uplo,
                                    armpl_int_t n, armpl_int_t nrhs,
                                    armpl_singlecomplex_t *a, armpl_int_t lda,
                                    armpl_singlecomplex_t *tb,
```

(continues on next page)

(continued from previous page)

```
                                    armpl_int_t ltb, armpl_int_t *ipiv,
                                    armpl_int_t *ipiv2,
                                    armpl_singlecomplex_t *b,
                                    armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chesv_aa_2stage* and *LAPACKE_zhesv_aa_2stage*. It also exists with a native Fortran interface as *chesv_aa_2stage*.

4.19.232 LAPACKE_chesv_aa_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chesv_aa_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, armpl_int_t nrhs,
                                  armpl_singlecomplex_t *a, armpl_int_t lda,
                                  armpl_int_t *ipiv, armpl_singlecomplex_t *b,
                                  armpl_int_t ldb,
                                  armpl_singlecomplex_t *work,
                                  armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chesv_aa_work* and *LAPACKE_zhesv_aa_work*.

4.19.233 LAPACKE_chesv_rk

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chesv_rk(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, armpl_int_t nrhs,
                             armpl_singlecomplex_t *a, armpl_int_t lda,
                             armpl_singlecomplex_t *e, armpl_int_t *ipiv,
                             armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chesv_rk* and *LAPACKE_zhesv_rk*. It also exists with a native Fortran interface as *chesv_rk*.

4.19.234 LAPACKE_chesv_rk_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chesv_rk_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, armpl_int_t nrhs,
                                  armpl_singlecomplex_t *a, armpl_int_t lda,
                                  armpl_singlecomplex_t *e, armpl_int_t *ipiv,
                                  armpl_singlecomplex_t *b, armpl_int_t ldb,
                                  armpl_singlecomplex_t *work,
                                  armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chesv_rk_work* and *LAPACKE_zhesv_rk_work*.

4.19.235 LAPACKE_chesv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chesv_work(armpl_int_t matrix_layout, char uplo,
                               armpl_int_t n, armpl_int_t nrhs,
                               armpl_singlecomplex_t *a, armpl_int_t lda,
                               armpl_int_t *ipiv, armpl_singlecomplex_t *b,
                               armpl_int_t ldb, armpl_singlecomplex_t *work,
                               armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chesv_work* and *LAPACKE_zhesv_work*.

4.19.236 LAPACKE_chesvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chesvx(armpl_int_t matrix_layout, char fact, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_singlecomplex_t *a, armpl_int_t lda,
                           armpl_singlecomplex_t *af, armpl_int_t ldaf,
                           armpl_int_t *ipiv, const armpl_singlecomplex_t *b,
                           armpl_int_t ldb, armpl_singlecomplex_t *x,
                           armpl_int_t ldx, float *rcond, float *ferr,
                           float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chesvx* and *LAPACKE_zhesvx*. It also exists with a native Fortran interface as *chesvx*.

4.19.237 LAPACKE_chesvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chesvx_work(armpl_int_t matrix_layout, char fact,
                                char uplo, armpl_int_t n, armpl_int_t nrhs,
                                const armpl_singlecomplex_t *a,
                                armpl_int_t lda, armpl_singlecomplex_t *af,
                                armpl_int_t ldaf, armpl_int_t *ipiv,
                                const armpl_singlecomplex_t *b,
                                armpl_int_t ldb, armpl_singlecomplex_t *x,
                                armpl_int_t ldx, float *rcond, float *ferr,
                                float *berr, armpl_singlecomplex_t *work,
                                armpl_int_t lwork, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chesvx_work* and *LAPACKE_zhesvx_work*.

4.19.238 LAPACKE_chesvxx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chesvxx(armpl_int_t matrix_layout, char fact, char uplo,
                            armpl_int_t n, armpl_int_t nrhs,
                            armpl_singlecomplex_t *a, armpl_int_t lda,
                            armpl_singlecomplex_t *af, armpl_int_t ldaf,
                            armpl_int_t *ipiv, char *equed, float *s,
                            armpl_singlecomplex_t *b, armpl_int_t ldb,
                            armpl_singlecomplex_t *x, armpl_int_t ldx,
```

(continues on next page)

(continued from previous page)

```
                                float *rcond, float *rpvgrw, float *berr,
                                armpl_int_t n_err_bnds, float *err_bnds_norm,
                                float *err_bnds_comp, armpl_int_t nparams,
                                float *params);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chesvxx* and *LAPACKE_zhesvxx*. It also exists with a native Fortran interface as *chesvxx*.

4.19.239 LAPACKE_chesvxx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chesvxx_work(armpl_int_t matrix_layout, char fact,
                                 char uplo, armpl_int_t n, armpl_int_t nrhs,
                                 armpl_singlecomplex_t *a, armpl_int_t lda,
                                 armpl_singlecomplex_t *af, armpl_int_t ldaf,
                                 armpl_int_t *ipiv, char *equed, float *s,
                                 armpl_singlecomplex_t *b, armpl_int_t ldb,
                                 armpl_singlecomplex_t *x, armpl_int_t ldx,
                                 float *rcond, float *rpvgrw, float *berr,
                                 armpl_int_t n_err_bnds, float *err_bnds_norm,
                                 float *err_bnds_comp, armpl_int_t nparams,
                                 float *params, armpl_singlecomplex_t *work,
                                 float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chesvxx_work* and *LAPACKE_zhesvxx_work*.

4.19.240 LAPACKE_cheswapr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cheswapr(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, armpl_singlecomplex_t *a,
                             armpl_int_t lda, armpl_int_t i1,
                             armpl_int_t i2);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheswapr* and *LAPACKE_zheswapr*. It also exists with a native Fortran interface as *cheswapr*.

4.19.241 LAPACKE_cheswapr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cheswapr_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, armpl_singlecomplex_t *a,
                                  armpl_int_t lda, armpl_int_t i1,
                                  armpl_int_t i2);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheswapr_work* and *LAPACKE_zheswapr_work*.

4.19.242 LAPACKE_chetrd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chetrd(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_singlecomplex_t *a,
                           armpl_int_t lda, float *d, float *e,
                           armpl_singlecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrd* and *LAPACKE_zhetrd*. It also exists with a native Fortran interface as *chetrd*.

4.19.243 LAPACKE_chetrd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chetrd_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_singlecomplex_t *a,
                                armpl_int_t lda, float *d, float *e,
                                armpl_singlecomplex_t *tau,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrd_work* and *LAPACKE_zhetrd_work*.

4.19.244 LAPACKE_chetrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chetrf(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_singlecomplex_t *a,
                           armpl_int_t lda, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrf* and *LAPACKE_zhetrf*. It also exists with a native Fortran interface as *chetrf*.

4.19.245 LAPACKE_chetrf_aa

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chetrf_aa(armpl_int_t matrix_layout, char uplo,
                              armpl_int_t n, armpl_singlecomplex_t *a,
                              armpl_int_t lda, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrf_aa* and *LAPACKE_zhetrf_aa*. It also exists with a native Fortran interface as *chetrf_aa*.

4.19.246 LAPACKE_chetrf_aa_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chetrf_aa_2stage(armpl_int_t matrix_layout, char uplo,
                                     armpl_int_t n, armpl_singlecomplex_t *a,
                                     armpl_int_t lda,
                                     armpl_singlecomplex_t *tb,
                                     armpl_int_t ltb, armpl_int_t *ipiv,
                                     armpl_int_t *ipiv2);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrf_aa_2stage* and *LAPACKE_zhetrf_aa_2stage*. It also exists with a native Fortran interface as *chetrf_aa_2stage*.

4.19.247 LAPACKE_chetrf_aa_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chetrf_aa_work(armpl_int_t matrix_layout, char uplo,
                                   armpl_int_t n, armpl_singlecomplex_t *a,
                                   armpl_int_t lda, armpl_int_t *ipiv,
                                   armpl_singlecomplex_t *work,
                                   armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrf_aa_work* and *LAPACKE_zhetrf_aa_work*.

4.19.248 LAPACKE_chetrf_rk

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chetrf_rk(armpl_int_t matrix_layout, char uplo,
                              armpl_int_t n, armpl_singlecomplex_t *a,
                              armpl_int_t lda, armpl_singlecomplex_t *e,
                              armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrf_rk* and *LAPACKE_zhetrf_rk*. It also exists with a native Fortran interface as *chetrf_rk*.

4.19.249 LAPACKE_chetrf_rk_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chetrf_rk_work(armpl_int_t matrix_layout, char uplo,
                                   armpl_int_t n, armpl_singlecomplex_t *a,
                                   armpl_int_t lda, armpl_singlecomplex_t *e,
                                   armpl_int_t *ipiv,
                                   armpl_singlecomplex_t *work,
                                   armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrf_rk_work* and *LAPACKE_zhetrf_rk_work*.

4.19.250 LAPACKE_chetrf_rook

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chetrf_rook(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_singlecomplex_t *a,
                                armpl_int_t lda, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrf_rook* and *LAPACKE_zhetrf_rook*. It also exists with a native Fortran interface as *chetrf_rook*.

4.19.251 LAPACKE_chetrf_rook_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chetrf_rook_work(armpl_int_t matrix_layout, char uplo,
                                     armpl_int_t n, armpl_singlecomplex_t *a,
                                     armpl_int_t lda, armpl_int_t *ipiv,
                                     armpl_singlecomplex_t *work,
                                     armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrf_rook_work* and *LAPACKE_zhetrf_rook_work*.

4.19.252 LAPACKE_chetrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chetrf_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_singlecomplex_t *a,
                                armpl_int_t lda, armpl_int_t *ipiv,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrf_work* and *LAPACKE_zhetrf_work*.

4.19.253 LAPACKE_chetri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chetri(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_singlecomplex_t *a,
                           armpl_int_t lda, const armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetri* and *LAPACKE_zhetri*. It also exists with a native Fortran interface as *chetri*.

4.19.254 LAPACKE_chetri2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chetri2(armpl_int_t matrix_layout, char uplo,
                            armpl_int_t n, armpl_singlecomplex_t *a,
                            armpl_int_t lda, const armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetri2* and *LAPACKE_zhetri2*. It also exists with a native Fortran interface as *chetri2*.

4.19.255 LAPACKE_chetri2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chetri2_work(armpl_int_t matrix_layout, char uplo,
                                 armpl_int_t n, armpl_singlecomplex_t *a,
                                 armpl_int_t lda, const armpl_int_t *ipiv,
                                 armpl_singlecomplex_t *work,
                                 armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetri2_work* and *LAPACKE_zhetri2_work*.

4.19.256 LAPACKE_chetri2x

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chetri2x(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, armpl_singlecomplex_t *a,
                             armpl_int_t lda, const armpl_int_t *ipiv,
                             armpl_int_t nb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetri2x* and *LAPACKE_zhetri2x*. It also exists with a native Fortran interface as *chetri2x*.

4.19.257 LAPACKE_chetri2x_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chetri2x_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, armpl_singlecomplex_t *a,
                                  armpl_int_t lda, const armpl_int_t *ipiv,
                                  armpl_singlecomplex_t *work,
                                  armpl_int_t nb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetri2x_work* and *LAPACKE_zhetri2x_work*.

4.19.258 LAPACKE_chetri_3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chetri_3(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, armpl_singlecomplex_t *a,
                             armpl_int_t lda, const armpl_singlecomplex_t *e,
                             const armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetri_3* and *LAPACKE_zhetri_3*. It also exists with a native Fortran interface as *chetri_3*.

4.19.259 LAPACKE_chetri_3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chetri_3_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, armpl_singlecomplex_t *a,
                                  armpl_int_t lda,
                                  const armpl_singlecomplex_t *e,
                                  const armpl_int_t *ipiv,
                                  armpl_singlecomplex_t *work,
                                  armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetri_3_work* and *LAPACKE_zhetri_3_work*.

4.19.260 LAPACKE_chetri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chetri_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_singlecomplex_t *a,
                                armpl_int_t lda, const armpl_int_t *ipiv,
                                armpl_singlecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetri_work* and *LAPACKE_zhetri_work*.

4.19.261 LAPACKE_chetrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chetrs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_singlecomplex_t *a, armpl_int_t lda,
                           const armpl_int_t *ipiv, armpl_singlecomplex_t *b,
                           armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrs* and *LAPACKE_zhetrs*. It also exists with a native Fortran interface as *chetrs*.

4.19.262 LAPACKE_chetrs2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chetrs2(armpl_int_t matrix_layout, char uplo,
                            armpl_int_t n, armpl_int_t nrhs,
                            const armpl_singlecomplex_t *a, armpl_int_t lda,
                            const armpl_int_t *ipiv, armpl_singlecomplex_t *b,
                            armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrs2* and *LAPACKE_zhetrs2*. It also exists with a native Fortran interface as *chetrs2*.

4.19.263 LAPACKE_chetrs2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chetrs2_work(armpl_int_t matrix_layout, char uplo,
                                 armpl_int_t n, armpl_int_t nrhs,
                                 const armpl_singlecomplex_t *a,
                                 armpl_int_t lda, const armpl_int_t *ipiv,
                                 armpl_singlecomplex_t *b, armpl_int_t ldb,
                                 armpl_singlecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrs2_work* and *LAPACKE_zhetrs2_work*.

4.19.264 LAPACKE_chetrs_3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chetrs_3(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, armpl_int_t nrhs,
                             const armpl_singlecomplex_t *a, armpl_int_t lda,
                             const armpl_singlecomplex_t *e,
                             const armpl_int_t *ipiv,
                             armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrs_3* and *LAPACKE_zhetrs_3*. It also exists with a native Fortran interface as *chetrs_3*.

4.19.265 LAPACKE_chetrs_3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chetrs_3_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, armpl_int_t nrhs,
                                  const armpl_singlecomplex_t *a,
                                  armpl_int_t lda,
                                  const armpl_singlecomplex_t *e,
                                  const armpl_int_t *ipiv,
                                  armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrs_3_work* and *LAPACKE_zhetrs_3_work*.

4.19.266 LAPACKE_chetrs_aa

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chetrs_aa(armpl_int_t matrix_layout, char uplo,
                              armpl_int_t n, armpl_int_t nrhs,
                              const armpl_singlecomplex_t *a, armpl_int_t lda,
                              const armpl_int_t *ipiv,
                              armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrs_aa* and *LAPACKE_zhetrs_aa*. It also exists with a native Fortran interface as *chetrs_aa*.

4.19.267 LAPACKE_chetrs_aa_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chetrs_aa_2stage(armpl_int_t matrix_layout, char uplo,
                                     armpl_int_t n, armpl_int_t nrhs,
                                     armpl_singlecomplex_t *a,
                                     armpl_int_t lda,
                                     armpl_singlecomplex_t *tb,
                                     armpl_int_t ltb, armpl_int_t *ipiv,
                                     armpl_int_t *ipiv2,
                                     armpl_singlecomplex_t *b,
                                     armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrs_aa_2stage* and *LAPACKE_zhetrs_aa_2stage*. It also exists with a native Fortran interface as *chetrs_aa_2stage*.

4.19.268 LAPACKE_chetrs_aa_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chetrs_aa_work(armpl_int_t matrix_layout, char uplo,
                                   armpl_int_t n, armpl_int_t nrhs,
                                   const armpl_singlecomplex_t *a,
                                   armpl_int_t lda, const armpl_int_t *ipiv,
                                   armpl_singlecomplex_t *b, armpl_int_t ldb,
                                   armpl_singlecomplex_t *work,
                                   armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrs_aa_work* and *LAPACKE_zhetrs_aa_work*.

4.19.269 LAPACKE_chetrs_rook

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chetrs_rook(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const armpl_singlecomplex_t *a,
                                armpl_int_t lda, const armpl_int_t *ipiv,
                                armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrs_rook* and *LAPACKE_zhetrs_rook*. It also exists with a native Fortran interface as *chetrs_rook*.

4.19.270 LAPACKE_chetrs_rook_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chetrs_rook_work(armpl_int_t matrix_layout, char uplo,
                                     armpl_int_t n, armpl_int_t nrhs,
                                     const armpl_singlecomplex_t *a,
                                     armpl_int_t lda, const armpl_int_t *ipiv,
                                     armpl_singlecomplex_t *b,
                                     armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrs_rook_work* and *LAPACKE_zhetrs_rook_work*.

4.19.271 LAPACKE_chetrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chetrs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const armpl_singlecomplex_t *a,
                                armpl_int_t lda, const armpl_int_t *ipiv,
                                armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.272 LAPACKE_chfrk

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chfrk(armpl_int_t matrix_layout, char transr, char uplo,
                          char trans, armpl_int_t n, armpl_int_t k,
                          float alpha, const armpl_singlecomplex_t *a,
                          armpl_int_t lda, float beta,
                          armpl_singlecomplex_t *c);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chfrk* and *LAPACKE_zhfrk*. It also exists with a native Fortran interface as *chfrk*.

4.19.273 LAPACKE_chfrk_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chfrk_work(armpl_int_t matrix_layout, char transr,
                               char uplo, char trans, armpl_int_t n,
                               armpl_int_t k, float alpha,
                               const armpl_singlecomplex_t *a,
                               armpl_int_t lda, float beta,
                               armpl_singlecomplex_t *c);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chfrk_work* and *LAPACKE_zhfrk_work*.

4.19.274 LAPACKE_chgeqz

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chgeqz(armpl_int_t matrix_layout, char job, char compq,
                           char compz, armpl_int_t n, armpl_int_t ilo,
                           armpl_int_t ihi, armpl_singlecomplex_t *h,
                           armpl_int_t ldh, armpl_singlecomplex_t *t,
                           armpl_int_t ldt, armpl_singlecomplex_t *alpha,
                           armpl_singlecomplex_t *beta,
                           armpl_singlecomplex_t *q, armpl_int_t ldq,
                           armpl_singlecomplex_t *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chgeqz*, *LAPACKE_dhgeqz*, *LAPACKE_shgeqz* and *LAPACKE_zhgeqz*. It also exists with a native Fortran interface as *chgeqz*.

4.19.275 LAPACKE_chgeqz_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chgeqz_work(armpl_int_t matrix_layout, char job,
                                char compq, char compz, armpl_int_t n,
```

(continues on next page)

(continued from previous page)

```
                                armpl_int_t ilo, armpl_int_t ihi,
                                armpl_singlecomplex_t *h, armpl_int_t ldh,
                                armpl_singlecomplex_t *t, armpl_int_t ldt,
                                armpl_singlecomplex_t *alpha,
                                armpl_singlecomplex_t *beta,
                                armpl_singlecomplex_t *q, armpl_int_t ldq,
                                armpl_singlecomplex_t *z, armpl_int_t ldz,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chgeqz_work*, *LAPACKE_dhgeqz_work*, *LAPACKE_shgeqz_work* and *LAPACKE_zhgeqz_work*.

4.19.276 LAPACKE_chpcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chpcon(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, const armpl_singlecomplex_t *ap,
                           const armpl_int_t *ipiv, float anorm,
                           float *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpcon* and *LAPACKE_zhpcon*. It also exists with a native Fortran interface as *chpcon*.

4.19.277 LAPACKE_chpcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chpcon_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n,
                                const armpl_singlecomplex_t *ap,
                                const armpl_int_t *ipiv, float anorm,
                                float *rcond, armpl_singlecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpcon_work* and *LAPACKE_zhpcon_work*.

4.19.278 LAPACKE_chpev

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chpev(armpl_int_t matrix_layout, char jobz, char uplo,
                          armpl_int_t n, armpl_singlecomplex_t *ap, float *w,
                          armpl_singlecomplex_t *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpev* and *LAPACKE_zhpev*. It also exists with a native Fortran interface as *chpev*.

4.19.279 LAPACKE_chpev_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chpev_work(armpl_int_t matrix_layout, char jobz,
                               char uplo, armpl_int_t n,
                               armpl_singlecomplex_t *ap, float *w,
                               armpl_singlecomplex_t *z, armpl_int_t ldz,
                               armpl_singlecomplex_t *work, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpev_work* and *LAPACKE_zhpev_work*.

4.19.280 LAPACKE_chpevd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chpevd(armpl_int_t matrix_layout, char jobz, char uplo,
                           armpl_int_t n, armpl_singlecomplex_t *ap, float *w,
                           armpl_singlecomplex_t *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpevd* and *LAPACKE_zhpevd*. It also exists with a native Fortran interface as *chpevd*.

4.19.281 LAPACKE_chpevd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chpevd_work(armpl_int_t matrix_layout, char jobz,
                                char uplo, armpl_int_t n,
                                armpl_singlecomplex_t *ap, float *w,
                                armpl_singlecomplex_t *z, armpl_int_t ldz,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork, float *rwork,
                                armpl_int_t lrwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpevd_work* and *LAPACKE_zhpevd_work*.

4.19.282 LAPACKE_chpevx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chpevx(armpl_int_t matrix_layout, char jobz, char range,
                           char uplo, armpl_int_t n,
                           armpl_singlecomplex_t *ap, float vl, float vu,
                           armpl_int_t il, armpl_int_t iu, float abstol,
                           armpl_int_t *m, float *w, armpl_singlecomplex_t *z,
                           armpl_int_t ldz, armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.283 LAPACKE_chpevx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chpevx_work(armpl_int_t matrix_layout, char jobz,
                                char range, char uplo, armpl_int_t n,
                                armpl_singlecomplex_t *ap, float vl, float vu,
                                armpl_int_t il, armpl_int_t iu, float abstol,
                                armpl_int_t *m, float *w,
                                armpl_singlecomplex_t *z, armpl_int_t ldz,
                                armpl_singlecomplex_t *work, float *rwork,
                                armpl_int_t *iwork, armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpevx_work* and *LAPACKE_zhpevx_work*.

4.19.284 LAPACKE_chpgst

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chpgst(armpl_int_t matrix_layout, armpl_int_t itype,
                           char uplo, armpl_int_t n,
                           armpl_singlecomplex_t *ap,
                           const armpl_singlecomplex_t *bp);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpgst* and *LAPACKE_zhpgst*. It also exists with a native Fortran interface as *chpgst*.

4.19.285 LAPACKE_chpgst_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chpgst_work(armpl_int_t matrix_layout, armpl_int_t itype,
                                char uplo, armpl_int_t n,
                                armpl_singlecomplex_t *ap,
                                const armpl_singlecomplex_t *bp);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpgst_work* and *LAPACKE_zhpgst_work*.

4.19.286 LAPACKE_chpgv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chpgv(armpl_int_t matrix_layout, armpl_int_t itype,
                          char jobz, char uplo, armpl_int_t n,
                          armpl_singlecomplex_t *ap,
                          armpl_singlecomplex_t *bp, float *w,
                          armpl_singlecomplex_t *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpgv* and *LAPACKE_zhpgv*. It also exists with a native Fortran interface as *chpgv*.

4.19.287 LAPACKE_chpgv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chpgv_work(armpl_int_t matrix_layout, armpl_int_t itype,
                               char jobz, char uplo, armpl_int_t n,
                               armpl_singlecomplex_t *ap,
                               armpl_singlecomplex_t *bp, float *w,
                               armpl_singlecomplex_t *z, armpl_int_t ldz,
                               armpl_singlecomplex_t *work, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpgv_work* and *LAPACKE_zhpgv_work*.

4.19.288 LAPACKE_chpgvd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chpgvd(armpl_int_t matrix_layout, armpl_int_t itype,
                           char jobz, char uplo, armpl_int_t n,
                           armpl_singlecomplex_t *ap,
                           armpl_singlecomplex_t *bp, float *w,
                           armpl_singlecomplex_t *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpgvd* and *LAPACKE_zhpgvd*. It also exists with a native Fortran interface as *chpgvd*.

4.19.289 LAPACKE_chpgvd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chpgvd_work(armpl_int_t matrix_layout, armpl_int_t itype,
                                char jobz, char uplo, armpl_int_t n,
                                armpl_singlecomplex_t *ap,
                                armpl_singlecomplex_t *bp, float *w,
                                armpl_singlecomplex_t *z, armpl_int_t ldz,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork, float *rwork,
                                armpl_int_t lrwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpgvd_work* and *LAPACKE_zhpgvd_work*.

4.19.290 LAPACKE_chpgvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chpgvx(armpl_int_t matrix_layout, armpl_int_t itype,
                           char jobz, char range, char uplo, armpl_int_t n,
                           armpl_singlecomplex_t *ap,
                           armpl_singlecomplex_t *bp, float vl, float vu,
                           armpl_int_t il, armpl_int_t iu, float abstol,
                           armpl_int_t *m, float *w, armpl_singlecomplex_t *z,
                           armpl_int_t ldz, armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpgvx* and *LAPACKE_zhpgvx*. It also exists with a native Fortran interface as *chpgvx*.

4.19.291 LAPACKE_chpgvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chpgvx_work(armpl_int_t matrix_layout, armpl_int_t itype,
                                char jobz, char range, char uplo,
                                armpl_int_t n, armpl_singlecomplex_t *ap,
                                armpl_singlecomplex_t *bp, float vl, float vu,
                                armpl_int_t il, armpl_int_t iu, float abstol,
                                armpl_int_t *m, float *w,
                                armpl_singlecomplex_t *z, armpl_int_t ldz,
                                armpl_singlecomplex_t *work, float *rwork,
                                armpl_int_t *iwork, armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpgvx_work* and *LAPACKE_zhpgvx_work*.

4.19.292 LAPACKE_chprfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chprfs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_singlecomplex_t *ap,
                           const armpl_singlecomplex_t *afp,
                           const armpl_int_t *ipiv,
                           const armpl_singlecomplex_t *b, armpl_int_t ldb,
```

(continues on next page)

(continued from previous page)

```
                                armpl_singlecomplex_t *x, armpl_int_t ldx,
                                float *ferr, float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chprfs* and *LAPACKE_zhprfs*. It also exists with a native Fortran interface as *chprfs*.

4.19.293 LAPACKE_chprfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chprfs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const armpl_singlecomplex_t *ap,
                                const armpl_singlecomplex_t *afp,
                                const armpl_int_t *ipiv,
                                const armpl_singlecomplex_t *b,
                                armpl_int_t ldb, armpl_singlecomplex_t *x,
                                armpl_int_t ldx, float *ferr, float *berr,
                                armpl_singlecomplex_t *work, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chprfs_work* and *LAPACKE_zhprfs_work*.

4.19.294 LAPACKE_chpsv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chpsv(armpl_int_t matrix_layout, char uplo, armpl_int_t n,
                          armpl_int_t nrhs, armpl_singlecomplex_t *ap,
                          armpl_int_t *ipiv, armpl_singlecomplex_t *b,
                          armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpsv* and *LAPACKE_zhpsv*. It also exists with a native Fortran interface as *chpsv*.

4.19.295 LAPACKE_chpsv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chpsv_work(armpl_int_t matrix_layout, char uplo,
                               armpl_int_t n, armpl_int_t nrhs,
                               armpl_singlecomplex_t *ap, armpl_int_t *ipiv,
                               armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpsv_work* and *LAPACKE_zhpsv_work*.

4.19.296 LAPACKE_chpsvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chpsvx(armpl_int_t matrix_layout, char fact, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_singlecomplex_t *ap,
                           armpl_singlecomplex_t *afp, armpl_int_t *ipiv,
                           const armpl_singlecomplex_t *b, armpl_int_t ldb,
                           armpl_singlecomplex_t *x, armpl_int_t ldx,
                           float *rcond, float *ferr, float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpsvx* and *LAPACKE_zhpsvx*. It also exists with a native Fortran interface as *chpsvx*.

4.19.297 LAPACKE_chpsvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chpsvx_work(armpl_int_t matrix_layout, char fact,
                                char uplo, armpl_int_t n, armpl_int_t nrhs,
                                const armpl_singlecomplex_t *ap,
                                armpl_singlecomplex_t *afp, armpl_int_t *ipiv,
                                const armpl_singlecomplex_t *b,
                                armpl_int_t ldb, armpl_singlecomplex_t *x,
                                armpl_int_t ldx, float *rcond, float *ferr,
                                float *berr, armpl_singlecomplex_t *work,
                                float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpsvx_work* and *LAPACKE_zhpsvx_work*.

4.19.298 LAPACKE_chptrd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chptrd(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_singlecomplex_t *ap, float *d,
                           float *e, armpl_singlecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chptrd* and *LAPACKE_zhptrd*. It also exists with a native Fortran interface as *chptrd*.

4.19.299 LAPACKE_chptrd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chptrd_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_singlecomplex_t *ap,
                                float *d, float *e,
                                armpl_singlecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chptrd_work* and *LAPACKE_zhptrd_work*.

4.19.300 LAPACKE_chptrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chptrf(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_singlecomplex_t *ap,
                           armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chptrf* and *LAPACKE_zhptrf*. It also exists with a native Fortran interface as *chptrf*.

4.19.301 LAPACKE_chptrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chptrf_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_singlecomplex_t *ap,
                                armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chptrf_work* and *LAPACKE_zhptrf_work*.

4.19.302 LAPACKE_chptri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chptri(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_singlecomplex_t *ap,
                           const armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chptri* and *LAPACKE_zhptri*. It also exists with a native Fortran interface as *chptri*.

4.19.303 LAPACKE_chptri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chptri_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_singlecomplex_t *ap,
                                const armpl_int_t *ipiv,
                                armpl_singlecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chptri_work* and *LAPACKE_zhptri_work*.

4.19.304 LAPACKE_chptrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chptrs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_singlecomplex_t *ap,
                           const armpl_int_t *ipiv, armpl_singlecomplex_t *b,
                           armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chptrs* and *LAPACKE_zhptrs*. It also exists with a native Fortran interface as *chptrs*.

4.19.305 LAPACKE_chptrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chptrs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const armpl_singlecomplex_t *ap,
                                const armpl_int_t *ipiv,
                                armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chptrs_work* and *LAPACKE_zhptrs_work*.

4.19.306 LAPACKE_chsein

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chsein(armpl_int_t matrix_layout, char job, char eigsrc,
                           char initv, const armpl_int_t *select,
                           armpl_int_t n, const armpl_singlecomplex_t *h,
                           armpl_int_t ldh, armpl_singlecomplex_t *w,
                           armpl_singlecomplex_t *vl, armpl_int_t ldvl,
                           armpl_singlecomplex_t *vr, armpl_int_t ldvr,
                           armpl_int_t mm, armpl_int_t *m,
                           armpl_int_t *ifaill, armpl_int_t *ifailr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chsein*, *LAPACKE_dhsein*, *LAPACKE_shsein* and *LAPACKE_zhsein*. It also exists with a native Fortran interface as *chsein*.

4.19.307 LAPACKE_chsein_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chsein_work(armpl_int_t matrix_layout, char job,
                                char eigsrc, char initv,
                                const armpl_int_t *select, armpl_int_t n,
                                const armpl_singlecomplex_t *h,
                                armpl_int_t ldh, armpl_singlecomplex_t *w,
                                armpl_singlecomplex_t *vl, armpl_int_t ldvl,
                                armpl_singlecomplex_t *vr, armpl_int_t ldvr,
                                armpl_int_t mm, armpl_int_t *m,
                                armpl_singlecomplex_t *work, float *rwork,
                                armpl_int_t *ifaill, armpl_int_t *ifailr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chsein_work*, *LAPACKE_dhsein_work*, *LAPACKE_shsein_work* and *LAPACKE_zhsein_work*.

4.19.308 LAPACKE_chseqr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chseqr(armpl_int_t matrix_layout, char job, char compz,
                           armpl_int_t n, armpl_int_t ilo, armpl_int_t ihi,
                           armpl_singlecomplex_t *h, armpl_int_t ldh,
                           armpl_singlecomplex_t *w, armpl_singlecomplex_t *z,
                           armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chseqr*, *LAPACKE_dhseqr*, *LAPACKE_shseqr* and *LAPACKE_zhseqr*. It also exists with a native Fortran interface as *chseqr*.

4.19.309 LAPACKE_chseqr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_chseqr_work(armpl_int_t matrix_layout, char job,
                                char compz, armpl_int_t n, armpl_int_t ilo,
                                armpl_int_t ihi, armpl_singlecomplex_t *h,
                                armpl_int_t ldh, armpl_singlecomplex_t *w,
                                armpl_singlecomplex_t *z, armpl_int_t ldz,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chseqr_work*, *LAPACKE_dhseqr_work*, *LAPACKE_shseqr_work* and *LAPACKE_zhseqr_work*.

4.19.310 LAPACKE_clacgv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_clacgv(armpl_int_t n, armpl_singlecomplex_t *x,
                           armpl_int_t incx);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clacgv* and *LAPACKE_zlacgv*. It also exists with a native Fortran interface as *clacgv*.

4.19.311 LAPACKE_clacgv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_clacgv_work(armpl_int_t n, armpl_singlecomplex_t *x,
                                armpl_int_t incx);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clacgv_work* and *LAPACKE_zlacgv_work*.

4.19.312 LAPACKE_clacn2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_clacn2(armpl_int_t n, armpl_singlecomplex_t *v,
                           armpl_singlecomplex_t *x, float *est,
                           armpl_int_t *kase, armpl_int_t *isave);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clacn2*, *LAPACKE_dlacn2*, *LAPACKE_slacn2* and *LAPACKE_zlacn2*. It also exists with a native Fortran interface as *clacn2*.

4.19.313 LAPACKE_clacn2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_clacn2_work(armpl_int_t n, armpl_singlecomplex_t *v,
                                armpl_singlecomplex_t *x, float *est,
                                armpl_int_t *kase, armpl_int_t *isave);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clacn2_work*, *LAPACKE_dlacn2_work*, *LAPACKE_slacn2_work* and *LAPACKE_zlacn2_work*.

4.19.314 LAPACKE_clacp2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_clacp2(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t m, armpl_int_t n, const float *a,
                           armpl_int_t lda, armpl_singlecomplex_t *b,
                           armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clacp2* and *LAPACKE_zlacp2*. It also exists with a native Fortran interface as *clacp2*.

4.19.315 LAPACKE_clacp2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_clacp2_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t m, armpl_int_t n, const float *a,
                                armpl_int_t lda, armpl_singlecomplex_t *b,
                                armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clacp2_work* and *LAPACKE_zlacp2_work*.

4.19.316 LAPACKE_clacpy

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_clacpy(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t m, armpl_int_t n,
                           const armpl_singlecomplex_t *a, armpl_int_t lda,
                           armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clacpy*, *LAPACKE_dlacpy*, *LAPACKE_slacpy* and *LAPACKE_zlacpy*. It also exists with a native Fortran interface as *clacpy*.

4.19.317 LAPACKE_clacpy_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_clacpy_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t m, armpl_int_t n,
                                const armpl_singlecomplex_t *a,
                                armpl_int_t lda, armpl_singlecomplex_t *b,
                                armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clacpy_work*, *LAPACKE_dlacpy_work*, *LAPACKE_slacpy_work* and *LAPACKE_zlacpy_work*.

4.19.318 LAPACKE_clacrm

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_clacrm(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, const armpl_singlecomplex_t *a,
                           armpl_int_t lda, const float *b, armpl_int_t ldb,
                           armpl_singlecomplex_t *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clacrm* and *LAPACKE_zlacrm*. It also exists with a native Fortran interface as *clacrm*.

4.19.319 LAPACKE_clacrm_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_clacrm_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, const armpl_singlecomplex_t *a,
                                armpl_int_t lda, const float *b,
                                armpl_int_t ldb, armpl_singlecomplex_t *c,
                                armpl_int_t ldc, float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clacrm_work* and *LAPACKE_zlacrm_work*.

4.19.320 LAPACKE_clag2z

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_clag2z(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, const armpl_singlecomplex_t *sa,
                           armpl_int_t ldsa, armpl_doublecomplex_t *a,
                           armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clag2z*. It also exists with a native Fortran interface as *clag2z*.

4.19.321 LAPACKE_clag2z_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_clag2z_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n,
                                const armpl_singlecomplex_t *sa,
                                armpl_int_t ldsa, armpl_doublecomplex_t *a,
                                armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clag2z_work*.

4.19.322 LAPACKE_clagge

Syntax

Routine is deprecated. Please choose a suitable alternative.

4.19.323 LAPACKE_clagge_work

Syntax

Routine is deprecated. Please choose a suitable alternative.

4.19.324 LAPACKE_claghe

Syntax

Routine is deprecated. Please choose a suitable alternative.

4.19.325 LAPACKE_claghe_work

Syntax

Routine is deprecated. Please choose a suitable alternative.

4.19.326 LAPACKE_clagsy

Syntax

Routine is deprecated. Please choose a suitable alternative.

4.19.327 LAPACKE_clagsy_work

Syntax

Routine is deprecated. Please choose a suitable alternative.

4.19.328 LAPACKE_clange

Syntax

```
#include "armpl.h"

float LAPACKE_clange(armpl_int_t matrix_layout, char norm, armpl_int_t m,
                     armpl_int_t n, const armpl_singlecomplex_t *a,
                     armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clange*, *LAPACKE_dlange*, *LAPACKE_slange* and *LAPACKE_zlange*. It also exists with a native Fortran interface as *clange*.

4.19.329 LAPACKE_clange_work

Syntax

```
#include "armpl.h"

float LAPACKE_clange_work(armpl_int_t matrix_layout, char norm, armpl_int_t m,
                          armpl_int_t n, const armpl_singlecomplex_t *a,
                          armpl_int_t lda, float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clange_work*, *LAPACKE_dlange_work*, *LAPACKE_slange_work* and *LAPACKE_zlange_work*.

4.19.330 LAPACKE_clanhe

Syntax

```
#include "armpl.h"

float LAPACKE_clanhe(armpl_int_t matrix_layout, char norm, char uplo,
                     armpl_int_t n, const armpl_singlecomplex_t *a,
                     armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clanhe* and *LAPACKE_zlanhe*. It also exists with a native Fortran interface as *clanhe*.

4.19.331 LAPACKE_clanhe_work

Syntax

```
#include "armpl.h"

float LAPACKE_clanhe_work(armpl_int_t matrix_layout, char norm, char uplo,
                          armpl_int_t n, const armpl_singlecomplex_t *a,
                          armpl_int_t lda, float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clanhe_work* and *LAPACKE_zlanhe_work*.

4.19.332 LAPACKE_clansy

Syntax

```
#include "armpl.h"

float LAPACKE_clansy(armpl_int_t matrix_layout, char norm, char uplo,
                     armpl_int_t n, const armpl_singlecomplex_t *a,
                     armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clansy*, *LAPACKE_dlansy*, *LAPACKE_slansy* and *LAPACKE_zlansy*. It also exists with a native Fortran interface as *clansy*.

4.19.333 LAPACKE_clansy_work

Syntax

```
#include "armpl.h"

float LAPACKE_clansy_work(armpl_int_t matrix_layout, char norm, char uplo,
                          armpl_int_t n, const armpl_singlecomplex_t *a,
                          armpl_int_t lda, float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clansy_work*, *LAPACKE_dlansy_work*, *LAPACKE_slansy_work* and *LAPACKE_zlansy_work*.

4.19.334 LAPACKE_clantr

Syntax

```
#include "armpl.h"

float LAPACKE_clantr(armpl_int_t matrix_layout, char norm, char uplo,
                     char diag, armpl_int_t m, armpl_int_t n,
                     const armpl_singlecomplex_t *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clantr*, *LAPACKE_dlantr*, *LAPACKE_slantr* and *LAPACKE_zlantr*. It also exists with a native Fortran interface as *clantr*.

4.19.335 LAPACKE_clantr_work

Syntax

```
#include "armpl.h"

float LAPACKE_clantr_work(armpl_int_t matrix_layout, char norm, char uplo,
                          char diag, armpl_int_t m, armpl_int_t n,
                          const armpl_singlecomplex_t *a, armpl_int_t lda,
                          float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clantr_work*, *LAPACKE_dlantr_work*, *LAPACKE_slantr_work* and *LAPACKE_zlantr_work*.

4.19.336 LAPACKE_clapmr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_clapmr(armpl_int_t matrix_layout, armpl_int_t forwrd,
                           armpl_int_t m, armpl_int_t n,
                           armpl_singlecomplex_t *x, armpl_int_t ldx,
                           armpl_int_t *k);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clapmr*, *LAPACKE_dlapmr*, *LAPACKE_slapmr* and *LAPACKE_zlapmr*. It also exists with a native Fortran interface as *clapmr*.

4.19.337 LAPACKE_clapmr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_clapmr_work(armpl_int_t matrix_layout, armpl_int_t forwrd,
                                armpl_int_t m, armpl_int_t n,
                                armpl_singlecomplex_t *x, armpl_int_t ldx,
                                armpl_int_t *k);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clapmr_work*, *LAPACKE_dlapmr_work*, *LAPACKE_slapmr_work* and *LAPACKE_zlapmr_work*.

4.19.338 LAPACKE_clapmt

4.19.339 LAPACKE_clapmt_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_clapmt_work(armpl_int_t matrix_layout, armpl_int_t forwrd,
                                armpl_int_t m, armpl_int_t n,
                                armpl_singlecomplex_t *x, armpl_int_t ldx,
                                armpl_int_t *k);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clapmt_work*, *LAPACKE_dlapmt_work*, *LAPACKE_slapmt_work* and *LAPACKE_zlapmt_work*.

4.19.340 LAPACKE_clarcm

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_clarcm(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, const float *a, armpl_int_t lda,
                           const armpl_singlecomplex_t *b, armpl_int_t ldb,
                           armpl_singlecomplex_t *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarcm* and *LAPACKE_zlarcm*. It also exists with a native Fortran interface as *clarcm*.

4.19.341 LAPACKE_clarcm_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_clarcm_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, const float *a,
                                armpl_int_t lda,
                                const armpl_singlecomplex_t *b,
                                armpl_int_t ldb, armpl_singlecomplex_t *c,
                                armpl_int_t ldc, float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarcm_work* and *LAPACKE_zlarcm_work*.

4.19.342 LAPACKE_clarfb

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_clarfb(armpl_int_t matrix_layout, char side, char trans,
                           char direct, char storev, armpl_int_t m,
                           armpl_int_t n, armpl_int_t k,
                           const armpl_singlecomplex_t *v, armpl_int_t ldv,
                           const armpl_singlecomplex_t *t, armpl_int_t ldt,
                           armpl_singlecomplex_t *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarfb*, *LAPACKE_dlarfb*, *LAPACKE_slarfb* and *LAPACKE_zlarfb*. It also exists with a native Fortran interface as *clarfb*.

4.19.343 LAPACKE_clarfb_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_clarfb_work(armpl_int_t matrix_layout, char side,
                                char trans, char direct, char storev,
                                armpl_int_t m, armpl_int_t n, armpl_int_t k,
                                const armpl_singlecomplex_t *v,
                                armpl_int_t ldv,
                                const armpl_singlecomplex_t *t,
                                armpl_int_t ldt, armpl_singlecomplex_t *c,
                                armpl_int_t ldc, armpl_singlecomplex_t *work,
                                armpl_int_t ldwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarfb_work*, *LAPACKE_dlarfb_work*, *LAPACKE_slarfb_work* and *LAPACKE_zlarfb_work*.

4.19.344 LAPACKE_clarfg

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_clarfg(armpl_int_t n, armpl_singlecomplex_t *alpha,
                           armpl_singlecomplex_t *x, armpl_int_t incx,
                           armpl_singlecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarfg*, *LAPACKE_dlarfg*, *LAPACKE_slarfg* and *LAPACKE_zlarfg*. It also exists with a native Fortran interface as *clarfg*.

4.19.345 LAPACKE_clarfg_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_clarfg_work(armpl_int_t n, armpl_singlecomplex_t *alpha,
                                armpl_singlecomplex_t *x, armpl_int_t incx,
                                armpl_singlecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarfg_work*, *LAPACKE_dlarfg_work*, *LAPACKE_slarfg_work* and *LAPACKE_zlarfg_work*.

4.19.346 LAPACKE_clarft

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_clarft(armpl_int_t matrix_layout, char direct,
                           char storev, armpl_int_t n, armpl_int_t k,
                           const armpl_singlecomplex_t *v, armpl_int_t ldv,
                           const armpl_singlecomplex_t *tau,
                           armpl_singlecomplex_t *t, armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarft*, *LAPACKE_dlarft*, *LAPACKE_slarft* and *LAPACKE_zlarft*. It also exists with a native Fortran interface as *clarft*.

4.19.347 LAPACKE_clarft_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_clarft_work(armpl_int_t matrix_layout, char direct,
                                char storev, armpl_int_t n, armpl_int_t k,
                                const armpl_singlecomplex_t *v,
                                armpl_int_t ldv,
                                const armpl_singlecomplex_t *tau,
                                armpl_singlecomplex_t *t, armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarft_work*, *LAPACKE_dlarft_work*, *LAPACKE_slarft_work* and *LAPACKE_zlarft_work*.

4.19.348 LAPACKE_clarfx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_clarfx(armpl_int_t matrix_layout, char side,
                           armpl_int_t m, armpl_int_t n,
                           const armpl_singlecomplex_t *v,
                           armpl_singlecomplex_t tau,
                           armpl_singlecomplex_t *c, armpl_int_t ldc,
                           armpl_singlecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarfx*, *LAPACKE_dlarfx*, *LAPACKE_slarfx* and *LAPACKE_zlarfx*. It also exists with a native Fortran interface as *clarfx*.

4.19.349 LAPACKE_clarfx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_clarfx_work(armpl_int_t matrix_layout, char side,
                                armpl_int_t m, armpl_int_t n,
                                const armpl_singlecomplex_t *v,
                                armpl_singlecomplex_t tau,
                                armpl_singlecomplex_t *c, armpl_int_t ldc,
                                armpl_singlecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarfx_work*, *LAPACKE_dlarfx_work*, *LAPACKE_slarfx_work* and *LAPACKE_zlarfx_work*.

4.19.350 LAPACKE_clarnv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_clarnv(armpl_int_t idist, armpl_int_t *iseed,
                           armpl_int_t n, armpl_singlecomplex_t *x);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarnv*, *LAPACKE_dlarnv*, *LAPACKE_slarnv* and *LAPACKE_zlarnv*. It also exists with a native Fortran interface as *clarnv*.

4.19.351 LAPACKE_clarnv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_clarnv_work(armpl_int_t idist, armpl_int_t *iseed,
                                armpl_int_t n, armpl_singlecomplex_t *x);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarnv_work*, *LAPACKE_dlarnv_work*, *LAPACKE_slarnv_work* and *LAPACKE_zlarnv_work*.

4.19.352 LAPACKE_clascl

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_clascl(armpl_int_t matrix_layout, char type,
                           armpl_int_t kl, armpl_int_t ku, float cfrom,
                           float cto, armpl_int_t m, armpl_int_t n,
                           armpl_singlecomplex_t *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clascl*, *LAPACKE_dlascl*, *LAPACKE_slascl* and *LAPACKE_zlascl*. It also exists with a native Fortran interface as *clascl*.

4.19.353 LAPACKE_clascl_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_clascl_work(armpl_int_t matrix_layout, char type,
                                armpl_int_t kl, armpl_int_t ku, float cfrom,
                                float cto, armpl_int_t m, armpl_int_t n,
                                armpl_singlecomplex_t *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clascl_work*, *LAPACKE_dlascl_work*, *LAPACKE_slascl_work* and *LAPACKE_zlascl_work*.

4.19.354 LAPACKE_claset

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_claset(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t m, armpl_int_t n,
                           armpl_singlecomplex_t alpha,
                           armpl_singlecomplex_t beta,
                           armpl_singlecomplex_t *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_claset*, *LAPACKE_dlaset*, *LAPACKE_slaset* and *LAPACKE_zlaset*. It also exists with a native Fortran interface as *claset*.

4.19.355 LAPACKE_claset_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_claset_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t m, armpl_int_t n,
                                armpl_singlecomplex_t alpha,
                                armpl_singlecomplex_t beta,
                                armpl_singlecomplex_t *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_claset_work*, *LAPACKE_dlaset_work*, *LAPACKE_slaset_work* and *LAPACKE_zlaset_work*.

4.19.356 LAPACKE_classq

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_classq(armpl_int_t n, armpl_singlecomplex_t *x,
                           armpl_int_t incx, float *scale, float *sumsq);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_classq*, *LAPACKE_dlassq*, *LAPACKE_slassq* and *LAPACKE_zlassq*. It also exists with a native Fortran interface as *classq*.

4.19.357 LAPACKE_classq_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_classq_work(armpl_int_t n, armpl_singlecomplex_t *x,
                                armpl_int_t incx, float *scale,
                                float *sumsq);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_classq_work*, *LAPACKE_dlassq_work*, *LAPACKE_slassq_work* and *LAPACKE_zlassq_work*.

4.19.358 LAPACKE_claswp

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_claswp(armpl_int_t matrix_layout, armpl_int_t n,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           armpl_int_t k1, armpl_int_t k2,
                           const armpl_int_t *ipiv, armpl_int_t incx);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_claswp*, *LAPACKE_dlaswp*, *LAPACKE_slaswp* and *LAPACKE_zlaswp*. It also exists with a native Fortran interface as *claswp*.

4.19.359 LAPACKE_claswp_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_claswp_work(armpl_int_t matrix_layout, armpl_int_t n,
                                armpl_singlecomplex_t *a, armpl_int_t lda,
                                armpl_int_t k1, armpl_int_t k2,
                                const armpl_int_t *ipiv, armpl_int_t incx);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_claswp_work*, *LAPACKE_dlaswp_work*, *LAPACKE_slaswp_work* and *LAPACKE_zlaswp_work*.

4.19.360 LAPACKE_clatms

Syntax

Routine is deprecated. Please choose a suitable alternative.

4.19.361 LAPACKE_clatms_work

Syntax

Routine is deprecated. Please choose a suitable alternative.

4.19.362 LAPACKE_clauum

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_clauum(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_singlecomplex_t *a,
                           armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clauum*, *LAPACKE_dlauum*, *LAPACKE_slauum* and *LAPACKE_zlauum*. It also exists with a native Fortran interface as *clauum*.

4.19.363 LAPACKE_clauum_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_clauum_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_singlecomplex_t *a,
                                armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clauum_work*, *LAPACKE_dlauum_work*, *LAPACKE_slauum_work* and *LAPACKE_zlauum_work*.

4.19.364 LAPACKE_cpbcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpbcon(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t kd,
                           const armpl_singlecomplex_t *ab, armpl_int_t ldab,
                           float anorm, float *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbcon*, *LAPACKE_dpbcon*, *LAPACKE_spbcon* and *LAPACKE_zpbcon*. It also exists with a native Fortran interface as *cpbcon*.

4.19.365 LAPACKE_cpbcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpbcon_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t kd,
                                const armpl_singlecomplex_t *ab,
                                armpl_int_t ldab, float anorm, float *rcond,
                                armpl_singlecomplex_t *work, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbcon_work*, *LAPACKE_dpbcon_work*, *LAPACKE_spbcon_work* and *LAPACKE_zpbcon_work*.

4.19.366 LAPACKE_cpbequ

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpbequ(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t kd,
                           const armpl_singlecomplex_t *ab, armpl_int_t ldab,
                           float *s, float *scond, float *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbequ*, *LAPACKE_dpbequ*, *LAPACKE_spbequ* and *LAPACKE_zpbequ*. It also exists with a native Fortran interface as *cpbequ*.

4.19.367 LAPACKE_cpbequ_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpbequ_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t kd,
                                const armpl_singlecomplex_t *ab,
                                armpl_int_t ldab, float *s, float *scond,
                                float *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbequ_work*, *LAPACKE_dpbequ_work*, *LAPACKE_spbequ_work* and *LAPACKE_zpbequ_work*.

4.19.368 LAPACKE_cpbrfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpbrfs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t kd, armpl_int_t nrhs,
                           const armpl_singlecomplex_t *ab, armpl_int_t ldab,
                           const armpl_singlecomplex_t *afb,
                           armpl_int_t ldafb, const armpl_singlecomplex_t *b,
                           armpl_int_t ldb, armpl_singlecomplex_t *x,
                           armpl_int_t ldx, float *ferr, float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbrfs*, *LAPACKE_dpbrfs*, *LAPACKE_spbrfs* and *LAPACKE_zpbrfs*. It also exists with a native Fortran interface as *cpbrfs*.

4.19.369 LAPACKE_cpbrfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpbrfs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t kd,
                                armpl_int_t nrhs,
                                const armpl_singlecomplex_t *ab,
                                armpl_int_t ldab,
                                const armpl_singlecomplex_t *afb,
                                armpl_int_t ldafb,
                                const armpl_singlecomplex_t *b,
                                armpl_int_t ldb, armpl_singlecomplex_t *x,
                                armpl_int_t ldx, float *ferr, float *berr,
                                armpl_singlecomplex_t *work, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbrfs_work*, *LAPACKE_dpbrfs_work*, *LAPACKE_spbrfs_work* and *LAPACKE_zpbrfs_work*.

4.19.370 LAPACKE_cpbstf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpbstf(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t kb,
                           armpl_singlecomplex_t *bb, armpl_int_t ldbb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbstf*, *LAPACKE_dpbstf*, *LAPACKE_spbstf* and *LAPACKE_zpbstf*. It also exists with a native Fortran interface as *cpbstf*.

4.19.371 LAPACKE_cpbstf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpbstf_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t kb,
                                armpl_singlecomplex_t *bb, armpl_int_t ldbb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbstf_work*, *LAPACKE_dpbstf_work*, *LAPACKE_spbstf_work* and *LAPACKE_zpbstf_work*.

4.19.372 LAPACKE_cpbsv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpbsv(armpl_int_t matrix_layout, char uplo, armpl_int_t n,
                          armpl_int_t kd, armpl_int_t nrhs,
                          armpl_singlecomplex_t *ab, armpl_int_t ldab,
                          armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbsv*, *LAPACKE_dpbsv*, *LAPACKE_spbsv* and *LAPACKE_zpbsv*. It also exists with a native Fortran interface as *cpbsv*.

4.19.373 LAPACKE_cpbsv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpbsv_work(armpl_int_t matrix_layout, char uplo,
                               armpl_int_t n, armpl_int_t kd,
                               armpl_int_t nrhs, armpl_singlecomplex_t *ab,
                               armpl_int_t ldab, armpl_singlecomplex_t *b,
                               armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbsv_work*, *LAPACKE_dpbsv_work*, *LAPACKE_spbsv_work* and *LAPACKE_zpbsv_work*.

4.19.374 LAPACKE_cpbsvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpbsvx(armpl_int_t matrix_layout, char fact, char uplo,
                           armpl_int_t n, armpl_int_t kd, armpl_int_t nrhs,
                           armpl_singlecomplex_t *ab, armpl_int_t ldab,
                           armpl_singlecomplex_t *afb, armpl_int_t ldafb,
                           char *equed, float *s, armpl_singlecomplex_t *b,
                           armpl_int_t ldb, armpl_singlecomplex_t *x,
                           armpl_int_t ldx, float *rcond, float *ferr,
                           float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbsvx*, *LAPACKE_dpbsvx*, *LAPACKE_spbsvx* and *LAPACKE_zpbsvx*. It also exists with a native Fortran interface as *cpbsvx*.

4.19.375 LAPACKE_cpbsvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpbsvx_work(armpl_int_t matrix_layout, char fact,
                                char uplo, armpl_int_t n, armpl_int_t kd,
                                armpl_int_t nrhs, armpl_singlecomplex_t *ab,
                                armpl_int_t ldab, armpl_singlecomplex_t *afb,
                                armpl_int_t ldafb, char *equed, float *s,
                                armpl_singlecomplex_t *b, armpl_int_t ldb,
                                armpl_singlecomplex_t *x, armpl_int_t ldx,
```

(continues on next page)

(continued from previous page)

```
                                    float *rcond, float *ferr, float *berr,
                                    armpl_singlecomplex_t *work, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbsvx_work*, *LAPACKE_dpbsvx_work*, *LAPACKE_spbsvx_work* and *LAPACKE_zpbsvx_work*.

4.19.376 LAPACKE_cpbtrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpbtrf(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t kd,
                           armpl_singlecomplex_t *ab, armpl_int_t ldab);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbtrf*, *LAPACKE_dpbtrf*, *LAPACKE_spbtrf* and *LAPACKE_zpbtrf*. It also exists with a native Fortran interface as *cpbtrf*.

4.19.377 LAPACKE_cpbtrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpbtrf_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t kd,
                                armpl_singlecomplex_t *ab, armpl_int_t ldab);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbtrf_work*, *LAPACKE_dpbtrf_work*, *LAPACKE_spbtrf_work* and *LAPACKE_zpbtrf_work*.

4.19.378 LAPACKE_cpbtrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpbtrs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t kd, armpl_int_t nrhs,
                           const armpl_singlecomplex_t *ab, armpl_int_t ldab,
                           armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbtrs*, *LAPACKE_dpbtrs*, *LAPACKE_spbtrs* and *LAPACKE_zpbtrs*. It also exists with a native Fortran interface as *cpbtrs*.

4.19.379 LAPACKE_cpbtrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpbtrs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t kd,
                                armpl_int_t nrhs,
                                const armpl_singlecomplex_t *ab,
                                armpl_int_t ldab, armpl_singlecomplex_t *b,
                                armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbtrs_work*, *LAPACKE_dpbtrs_work*, *LAPACKE_spbtrs_work* and *LAPACKE_zpbtrs_work*.

4.19.380 LAPACKE_cpftrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpftrf(armpl_int_t matrix_layout, char transr, char uplo,
                           armpl_int_t n, armpl_singlecomplex_t *a);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpftrf*, *LAPACKE_dpftrf*, *LAPACKE_spftrf* and *LAPACKE_zpftrf*. It also exists with a native Fortran interface as *cpftrf*.

4.19.381 LAPACKE_cpftrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpftrf_work(armpl_int_t matrix_layout, char transr,
                                char uplo, armpl_int_t n,
                                armpl_singlecomplex_t *a);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpftrf_work*, *LAPACKE_dpftrf_work*, *LAPACKE_spftrf_work* and *LAPACKE_zpftrf_work*.

4.19.382 LAPACKE_cpftri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpftri(armpl_int_t matrix_layout, char transr, char uplo,
                           armpl_int_t n, armpl_singlecomplex_t *a);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpftri*, *LAPACKE_dpftri*, *LAPACKE_spftri* and *LAPACKE_zpftri*. It also exists with a native Fortran interface as *cpftri*.

4.19.383 LAPACKE_cpftri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpftri_work(armpl_int_t matrix_layout, char transr,
                                char uplo, armpl_int_t n,
                                armpl_singlecomplex_t *a);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpftri_work*, *LAPACKE_dpftri_work*, *LAPACKE_spftri_work* and *LAPACKE_zpftri_work*.

4.19.384 LAPACKE_cpftrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpftrs(armpl_int_t matrix_layout, char transr, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_singlecomplex_t *a,
                           armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpftrs*, *LAPACKE_dpftrs*, *LAPACKE_spftrs* and *LAPACKE_zpftrs*. It also exists with a native Fortran interface as *cpftrs*.

4.19.385 LAPACKE_cpftrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpftrs_work(armpl_int_t matrix_layout, char transr,
                                char uplo, armpl_int_t n, armpl_int_t nrhs,
                                const armpl_singlecomplex_t *a,
                                armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpftrs_work*, *LAPACKE_dpftrs_work*, *LAPACKE_spftrs_work* and *LAPACKE_zpftrs_work*.

4.19.386 LAPACKE_cpocon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpocon(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, const armpl_singlecomplex_t *a,
                           armpl_int_t lda, float anorm, float *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.387 LAPACKE_cpocon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpocon_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, const armpl_singlecomplex_t *a,
                                armpl_int_t lda, float anorm, float *rcond,
                                armpl_singlecomplex_t *work, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpocon_work*, *LAPACKE_dpocon_work*, *LAPACKE_spocon_work* and *LAPACKE_zpocon_work*.

4.19.388 LAPACKE_cpoequ

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpoequ(armpl_int_t matrix_layout, armpl_int_t n,
                           const armpl_singlecomplex_t *a, armpl_int_t lda,
                           float *s, float *scond, float *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpoequ*, *LAPACKE_dpoequ*, *LAPACKE_spoequ* and *LAPACKE_zpoequ*. It also exists with a native Fortran interface as *cpoequ*.

4.19.389 LAPACKE_cpoequ_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpoequ_work(armpl_int_t matrix_layout, armpl_int_t n,
                                const armpl_singlecomplex_t *a,
                                armpl_int_t lda, float *s, float *scond,
                                float *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.390 LAPACKE_cpoequb

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpoequb(armpl_int_t matrix_layout, armpl_int_t n,
                            const armpl_singlecomplex_t *a, armpl_int_t lda,
                            float *s, float *scond, float *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpoequb*, *LAPACKE_dpoequb*, *LAPACKE_spoequb* and *LAPACKE_zpoequb*. It also exists with a native Fortran interface as *cpoequb*.

4.19.391 LAPACKE_cpoequb_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpoequb_work(armpl_int_t matrix_layout, armpl_int_t n,
                                 const armpl_singlecomplex_t *a,
                                 armpl_int_t lda, float *s, float *scond,
                                 float *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpoequb_work*, *LAPACKE_dpoequb_work*, *LAPACKE_spoequb_work* and *LAPACKE_zpoequb_work*.

4.19.392 LAPACKE_cporfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cporfs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_singlecomplex_t *a, armpl_int_t lda,
                           const armpl_singlecomplex_t *af, armpl_int_t ldaf,
                           const armpl_singlecomplex_t *b, armpl_int_t ldb,
                           armpl_singlecomplex_t *x, armpl_int_t ldx,
                           float *ferr, float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cporfs*, *LAPACKE_dporfs*, *LAPACKE_sporfs* and *LAPACKE_zporfs*. It also exists with a native Fortran interface as *cporfs*.

4.19.393 LAPACKE_cporfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cporfs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const armpl_singlecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_singlecomplex_t *af,
                                armpl_int_t ldaf,
                                const armpl_singlecomplex_t *b,
                                armpl_int_t ldb, armpl_singlecomplex_t *x,
                                armpl_int_t ldx, float *ferr, float *berr,
                                armpl_singlecomplex_t *work, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cporfs_work*, *LAPACKE_dporfs_work*, *LAPACKE_sporfs_work* and *LAPACKE_zporfs_work*.

4.19.394 LAPACKE_cporfsx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cporfsx(armpl_int_t matrix_layout, char uplo, char equed,
                            armpl_int_t n, armpl_int_t nrhs,
                            const armpl_singlecomplex_t *a, armpl_int_t lda,
                            const armpl_singlecomplex_t *af, armpl_int_t ldaf,
                            const float *s, const armpl_singlecomplex_t *b,
                            armpl_int_t ldb, armpl_singlecomplex_t *x,
                            armpl_int_t ldx, float *rcond, float *berr,
                            armpl_int_t n_err_bnds, float *err_bnds_norm,
                            float *err_bnds_comp, armpl_int_t nparams,
                            float *params);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cporfsx*, *LAPACKE_dporfsx*, *LAPACKE_sporfsx* and *LAPACKE_zporfsx*. It also exists with a native Fortran interface as *cporfsx*.

4.19.395 LAPACKE_cporfsx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cporfsx_work(armpl_int_t matrix_layout, char uplo,
                                 char equed, armpl_int_t n, armpl_int_t nrhs,
                                 const armpl_singlecomplex_t *a,
                                 armpl_int_t lda,
                                 const armpl_singlecomplex_t *af,
                                 armpl_int_t ldaf, const float *s,
                                 const armpl_singlecomplex_t *b,
                                 armpl_int_t ldb, armpl_singlecomplex_t *x,
                                 armpl_int_t ldx, float *rcond, float *berr,
                                 armpl_int_t n_err_bnds, float *err_bnds_norm,
                                 float *err_bnds_comp, armpl_int_t nparams,
                                 float *params, armpl_singlecomplex_t *work,
                                 float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cporfsx_work*, *LAPACKE_dporfsx_work*, *LAPACKE_sporfsx_work* and *LAPACKE_zporfsx_work*.

4.19.396 LAPACKE_cposv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cposv(armpl_int_t matrix_layout, char uplo, armpl_int_t n,
                          armpl_int_t nrhs, armpl_singlecomplex_t *a,
                          armpl_int_t lda, armpl_singlecomplex_t *b,
                          armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cposv*, *LAPACKE_dposv*, *LAPACKE_sposv* and *LAPACKE_zposv*. It also exists with a native Fortran interface as *cposv*.

4.19.397 LAPACKE_cposv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cposv_work(armpl_int_t matrix_layout, char uplo,
                               armpl_int_t n, armpl_int_t nrhs,
                               armpl_singlecomplex_t *a, armpl_int_t lda,
                               armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cposv_work*, *LAPACKE_dposv_work*, *LAPACKE_sposv_work* and *LAPACKE_zposv_work*.

4.19.398 LAPACKE_cposvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cposvx(armpl_int_t matrix_layout, char fact, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           armpl_singlecomplex_t *af, armpl_int_t ldaf,
                           char *equed, float *s, armpl_singlecomplex_t *b,
                           armpl_int_t ldb, armpl_singlecomplex_t *x,
                           armpl_int_t ldx, float *rcond, float *ferr,
                           float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cposvx*, *LAPACKE_dposvx*, *LAPACKE_sposvx* and *LAPACKE_zposvx*. It also exists with a native Fortran interface as *cposvx*.

4.19.399 LAPACKE_cposvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cposvx_work(armpl_int_t matrix_layout, char fact,
                                char uplo, armpl_int_t n, armpl_int_t nrhs,
                                armpl_singlecomplex_t *a, armpl_int_t lda,
                                armpl_singlecomplex_t *af, armpl_int_t ldaf,
                                char *equed, float *s,
                                armpl_singlecomplex_t *b, armpl_int_t ldb,
                                armpl_singlecomplex_t *x, armpl_int_t ldx,
                                float *rcond, float *ferr, float *berr,
                                armpl_singlecomplex_t *work, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cposvx_work*, *LAPACKE_dposvx_work*, *LAPACKE_sposvx_work* and *LAPACKE_zposvx_work*.

4.19.400 LAPACKE_cposvxx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cposvxx(armpl_int_t matrix_layout, char fact, char uplo,
                            armpl_int_t n, armpl_int_t nrhs,
                            armpl_singlecomplex_t *a, armpl_int_t lda,
                            armpl_singlecomplex_t *af, armpl_int_t ldaf,
                            char *equed, float *s, armpl_singlecomplex_t *b,
                            armpl_int_t ldb, armpl_singlecomplex_t *x,
                            armpl_int_t ldx, float *rcond, float *rpvgrw,
                            float *berr, armpl_int_t n_err_bnds,
                            float *err_bnds_norm, float *err_bnds_comp,
                            armpl_int_t nparams, float *params);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cposvxx*, *LAPACKE_dposvxx*, *LAPACKE_sposvxx* and *LAPACKE_zposvxx*. It also exists with a native Fortran interface as *cposvxx*.

4.19.401 LAPACKE_cposvxx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cposvxx_work(armpl_int_t matrix_layout, char fact,
                                 char uplo, armpl_int_t n, armpl_int_t nrhs,
                                 armpl_singlecomplex_t *a, armpl_int_t lda,
                                 armpl_singlecomplex_t *af, armpl_int_t ldaf,
                                 char *equed, float *s,
                                 armpl_singlecomplex_t *b, armpl_int_t ldb,
```

(continues on next page)

(continued from previous page)

```
                                armpl_singlecomplex_t *x, armpl_int_t ldx,
                                float *rcond, float *rpvgrw, float *berr,
                                armpl_int_t n_err_bnds, float *err_bnds_norm,
                                float *err_bnds_comp, armpl_int_t nparams,
                                float *params, armpl_singlecomplex_t *work,
                                float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cposvxx_work*, *LAPACKE_dposvxx_work*, *LAPACKE_sposvxx_work* and *LAPACKE_zposvxx_work*.

4.19.402 LAPACKE_cpotrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpotrf(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_singlecomplex_t *a,
                           armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpotrf*, *LAPACKE_dpotrf*, *LAPACKE_spotrf* and *LAPACKE_zpotrf*. It also exists with a native Fortran interface as *cpotrf*.

4.19.403 LAPACKE_cpotrf2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpotrf2(armpl_int_t matrix_layout, char uplo,
                            armpl_int_t n, armpl_singlecomplex_t *a,
                            armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpotrf2*, *LAPACKE_dpotrf2*, *LAPACKE_spotrf2* and *LAPACKE_zpotrf2*. It also exists with a native Fortran interface as *cpotrf2*.

4.19.404 LAPACKE_cpotrf2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpotrf2_work(armpl_int_t matrix_layout, char uplo,
                                 armpl_int_t n, armpl_singlecomplex_t *a,
                                 armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpotrf2_work*, *LAPACKE_dpotrf2_work*, *LAPACKE_spotrf2_work* and *LAPACKE_zpotrf2_work*.

4.19.405 LAPACKE_cpotrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpotrf_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_singlecomplex_t *a,
                                armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpotrf_work*, *LAPACKE_dpotrf_work*, *LAPACKE_spotrf_work* and *LAPACKE_zpotrf_work*.

4.19.406 LAPACKE_cpotri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpotri(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_singlecomplex_t *a,
                           armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpotri*, *LAPACKE_dpotri*, *LAPACKE_spotri* and *LAPACKE_zpotri*. It also exists with a native Fortran interface as *cpotri*.

4.19.407 LAPACKE_cpotri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpotri_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_singlecomplex_t *a,
                                armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpotri_work*, *LAPACKE_dpotri_work*, *LAPACKE_spotri_work* and *LAPACKE_zpotri_work*.

4.19.408 LAPACKE_cpotrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpotrs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_singlecomplex_t *a, armpl_int_t lda,
                           armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpotrs*, *LAPACKE_dpotrs*, *LAPACKE_spotrs* and *LAPACKE_zpotrs*. It also exists with a native Fortran interface as *cpotrs*.

4.19.409 LAPACKE_cpotrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpotrs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const armpl_singlecomplex_t *a,
                                armpl_int_t lda, armpl_singlecomplex_t *b,
                                armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpotrs_work*, *LAPACKE_dpotrs_work*, *LAPACKE_spotrs_work* and *LAPACKE_zpotrs_work*.

4.19.410 LAPACKE_cppcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cppcon(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, const armpl_singlecomplex_t *ap,
                           float anorm, float *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cppcon*, *LAPACKE_dppcon*, *LAPACKE_sppcon* and *LAPACKE_zppcon*. It also exists with a native Fortran interface as *cppcon*.

4.19.411 LAPACKE_cppcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cppcon_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n,
                                const armpl_singlecomplex_t *ap, float anorm,
                                float *rcond, armpl_singlecomplex_t *work,
                                float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cppcon_work*, *LAPACKE_dppcon_work*, *LAPACKE_sppcon_work* and *LAPACKE_zppcon_work*.

4.19.412 LAPACKE_cppequ

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cppequ(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, const armpl_singlecomplex_t *ap,
                           float *s, float *scond, float *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cppequ*, *LAPACKE_dppequ*, *LAPACKE_sppequ* and *LAPACKE_zppequ*. It also exists with a native Fortran interface as *cppequ*.

4.19.413 LAPACKE_cppequ_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cppequ_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n,
                                const armpl_singlecomplex_t *ap, float *s,
                                float *scond, float *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cppequ_work*, *LAPACKE_dppequ_work*, *LAPACKE_sppequ_work* and *LAPACKE_zppequ_work*.

4.19.414 LAPACKE_cpprfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpprfs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_singlecomplex_t *ap,
                           const armpl_singlecomplex_t *afp,
                           const armpl_singlecomplex_t *b, armpl_int_t ldb,
                           armpl_singlecomplex_t *x, armpl_int_t ldx,
                           float *ferr, float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpprfs*, *LAPACKE_dpprfs*, *LAPACKE_spprfs* and *LAPACKE_zpprfs*. It also exists with a native Fortran interface as *cpprfs*.

4.19.415 LAPACKE_cpprfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpprfs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const armpl_singlecomplex_t *ap,
                                const armpl_singlecomplex_t *afp,
                                const armpl_singlecomplex_t *b,
                                armpl_int_t ldb, armpl_singlecomplex_t *x,
                                armpl_int_t ldx, float *ferr, float *berr,
                                armpl_singlecomplex_t *work, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpprfs_work*, *LAPACKE_dpprfs_work*, *LAPACKE_spprfs_work* and *LAPACKE_zpprfs_work*.

4.19.416 LAPACKE_cppsv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cppsv(armpl_int_t matrix_layout, char uplo, armpl_int_t n,
                          armpl_int_t nrhs, armpl_singlecomplex_t *ap,
                          armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cppsv*, *LAPACKE_dppsv*, *LAPACKE_sppsv* and *LAPACKE_zppsv*. It also exists with a native Fortran interface as *cppsv*.

4.19.417 LAPACKE_cppsv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cppsv_work(armpl_int_t matrix_layout, char uplo,
                               armpl_int_t n, armpl_int_t nrhs,
                               armpl_singlecomplex_t *ap,
                               armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cppsv_work*, *LAPACKE_dppsv_work*, *LAPACKE_sppsv_work* and *LAPACKE_zppsv_work*.

4.19.418 LAPACKE_cppsvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cppsvx(armpl_int_t matrix_layout, char fact, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           armpl_singlecomplex_t *ap,
                           armpl_singlecomplex_t *afp, char *equed, float *s,
                           armpl_singlecomplex_t *b, armpl_int_t ldb,
                           armpl_singlecomplex_t *x, armpl_int_t ldx,
                           float *rcond, float *ferr, float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cppsvx*, *LAPACKE_dppsvx*, *LAPACKE_sppsvx* and *LAPACKE_zppsvx*. It also exists with a native Fortran interface as *cppsvx*.

4.19.419 LAPACKE_cppsvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cppsvx_work(armpl_int_t matrix_layout, char fact,
                                char uplo, armpl_int_t n, armpl_int_t nrhs,
                                armpl_singlecomplex_t *ap,
                                armpl_singlecomplex_t *afp, char *equed,
                                float *s, armpl_singlecomplex_t *b,
                                armpl_int_t ldb, armpl_singlecomplex_t *x,
                                armpl_int_t ldx, float *rcond, float *ferr,
                                float *berr, armpl_singlecomplex_t *work,
                                float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cppsvx_work*, *LAPACKE_dppsvx_work*, *LAPACKE_sppsvx_work* and *LAPACKE_zppsvx_work*.

4.19.420 LAPACKE_cpptrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpptrf(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_singlecomplex_t *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpptrf*, *LAPACKE_dpptrf*, *LAPACKE_spptrf* and *LAPACKE_zpptrf*. It also exists with a native Fortran interface as *cpptrf*.

4.19.421 LAPACKE_cpptrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpptrf_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_singlecomplex_t *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpptrf_work*, *LAPACKE_dpptrf_work*, *LAPACKE_spptrf_work* and *LAPACKE_zpptrf_work*.

4.19.422 LAPACKE_cpptri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpptri(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_singlecomplex_t *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpptri*, *LAPACKE_dpptri*, *LAPACKE_spptri* and *LAPACKE_zpptri*. It also exists with a native Fortran interface as *cpptri*.

4.19.423 LAPACKE_cpptri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpptri_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_singlecomplex_t *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpptri_work*, *LAPACKE_dpptri_work*, *LAPACKE_spptri_work* and *LAPACKE_zpptri_work*.

4.19.424 LAPACKE_cpptrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpptrs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_singlecomplex_t *ap,
                           armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpptrs*, *LAPACKE_dpptrs*, *LAPACKE_spptrs* and *LAPACKE_zpptrs*. It also exists with a native Fortran interface as *cpptrs*.

4.19.425 LAPACKE_cpptrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpptrs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const armpl_singlecomplex_t *ap,
                                armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpptrs_work*, *LAPACKE_dpptrs_work*, *LAPACKE_spptrs_work* and *LAPACKE_zpptrs_work*.

4.19.426 LAPACKE_cpstrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpstrf(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_singlecomplex_t *a,
                           armpl_int_t lda, armpl_int_t *piv,
                           armpl_int_t *rank, float tol);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpstrf*, *LAPACKE_dpstrf*, *LAPACKE_spstrf* and *LAPACKE_zpstrf*. It also exists with a native Fortran interface as *cpstrf*.

4.19.427 LAPACKE_cpstrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpstrf_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_singlecomplex_t *a,
                                armpl_int_t lda, armpl_int_t *piv,
                                armpl_int_t *rank, float tol, float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpstrf_work*, *LAPACKE_dpstrf_work*, *LAPACKE_spstrf_work* and *LAPACKE_zpstrf_work*.

4.19.428 LAPACKE_cptcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cptcon(armpl_int_t n, const float *d,
                           const armpl_singlecomplex_t *e, float anorm,
                           float *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cptcon*, *LAPACKE_dptcon*, *LAPACKE_sptcon* and *LAPACKE_zptcon*. It also exists with a native Fortran interface as *cptcon*.

4.19.429 LAPACKE_cptcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cptcon_work(armpl_int_t n, const float *d,
                                const armpl_singlecomplex_t *e, float anorm,
                                float *rcond, float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cptcon_work*, *LAPACKE_dptcon_work*, *LAPACKE_sptcon_work* and *LAPACKE_zptcon_work*.

4.19.430 LAPACKE_cpteqr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpteqr(armpl_int_t matrix_layout, char compz,
                           armpl_int_t n, float *d, float *e,
                           armpl_singlecomplex_t *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpteqr*, *LAPACKE_dpteqr*, *LAPACKE_spteqr* and *LAPACKE_zpteqr*. It also exists with a native Fortran interface as *cpteqr*.

4.19.431 LAPACKE_cpteqr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpteqr_work(armpl_int_t matrix_layout, char compz,
                                armpl_int_t n, float *d, float *e,
                                armpl_singlecomplex_t *z, armpl_int_t ldz,
                                float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpteqr_work*, *LAPACKE_dpteqr_work*, *LAPACKE_spteqr_work* and *LAPACKE_zpteqr_work*.

4.19.432 LAPACKE_cptrfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cptrfs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs, const float *d,
                           const armpl_singlecomplex_t *e, const float *df,
                           const armpl_singlecomplex_t *ef,
                           const armpl_singlecomplex_t *b, armpl_int_t ldb,
                           armpl_singlecomplex_t *x, armpl_int_t ldx,
                           float *ferr, float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cptrfs*, *LAPACKE_dptrfs*, *LAPACKE_sptrfs* and *LAPACKE_zptrfs*. It also exists with a native Fortran interface as *cptrfs*.

4.19.433 LAPACKE_cptrfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cptrfs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const float *d,
                                const armpl_singlecomplex_t *e,
                                const float *df,
                                const armpl_singlecomplex_t *ef,
                                const armpl_singlecomplex_t *b,
                                armpl_int_t ldb, armpl_singlecomplex_t *x,
                                armpl_int_t ldx, float *ferr, float *berr,
                                armpl_singlecomplex_t *work, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cptrfs_work*, *LAPACKE_dptrfs_work*, *LAPACKE_sptrfs_work* and *LAPACKE_zptrfs_work*.

4.19.434 LAPACKE_cptsv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cptsv(armpl_int_t matrix_layout, armpl_int_t n,
                          armpl_int_t nrhs, float *d,
                          armpl_singlecomplex_t *e, armpl_singlecomplex_t *b,
                          armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cptsv*, *LAPACKE_dptsv*, *LAPACKE_sptsv* and *LAPACKE_zptsv*. It also exists with a native Fortran interface as *cptsv*.

4.19.435 LAPACKE_cptsv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cptsv_work(armpl_int_t matrix_layout, armpl_int_t n,
                               armpl_int_t nrhs, float *d,
                               armpl_singlecomplex_t *e,
                               armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cptsv_work*, *LAPACKE_dptsv_work*, *LAPACKE_sptsv_work* and *LAPACKE_zptsv_work*.

4.19.436 LAPACKE_cptsvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cptsvx(armpl_int_t matrix_layout, char fact,
                           armpl_int_t n, armpl_int_t nrhs, const float *d,
                           const armpl_singlecomplex_t *e, float *df,
                           armpl_singlecomplex_t *ef,
                           const armpl_singlecomplex_t *b, armpl_int_t ldb,
                           armpl_singlecomplex_t *x, armpl_int_t ldx,
                           float *rcond, float *ferr, float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.437 LAPACKE_cptsvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cptsvx_work(armpl_int_t matrix_layout, char fact,
                                armpl_int_t n, armpl_int_t nrhs,
                                const float *d,
                                const armpl_singlecomplex_t *e, float *df,
                                armpl_singlecomplex_t *ef,
                                const armpl_singlecomplex_t *b,
                                armpl_int_t ldb, armpl_singlecomplex_t *x,
                                armpl_int_t ldx, float *rcond, float *ferr,
                                float *berr, armpl_singlecomplex_t *work,
                                float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cptsvx_work*, *LAPACKE_dptsvx_work*, *LAPACKE_sptsvx_work* and *LAPACKE_zptsvx_work*.

4.19.438 LAPACKE_cpttrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpttrf(armpl_int_t n, float *d,
                           armpl_singlecomplex_t *e);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpttrf*, *LAPACKE_dpttrf*, *LAPACKE_spttrf* and *LAPACKE_zpttrf*. It also exists with a native Fortran interface as *cpttrf*.

4.19.439 LAPACKE_cpttrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpttrf_work(armpl_int_t n, float *d,
                                armpl_singlecomplex_t *e);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpttrf_work*, *LAPACKE_dpttrf_work*, *LAPACKE_spttrf_work* and *LAPACKE_zpttrf_work*.

4.19.440 LAPACKE_cpttrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpttrs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs, const float *d,
                           const armpl_singlecomplex_t *e,
                           armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpttrs*, *LAPACKE_dpttrs*, *LAPACKE_spttrs* and *LAPACKE_zpttrs*. It also exists with a native Fortran interface as *cpttrs*.

4.19.441 LAPACKE_cpttrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cpttrs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const float *d,
                                const armpl_singlecomplex_t *e,
                                armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpttrs_work*, *LAPACKE_dpttrs_work*, *LAPACKE_spttrs_work* and *LAPACKE_zpttrs_work*.

4.19.442 LAPACKE_cspcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cspcon(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, const armpl_singlecomplex_t *ap,
                           const armpl_int_t *ipiv, float anorm,
                           float *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cspcon*, *LAPACKE_dspcon*, *LAPACKE_sspcon* and *LAPACKE_zspcon*. It also exists with a native Fortran interface as *cspcon*.

4.19.443 LAPACKE_cspcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cspcon_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n,
                                const armpl_singlecomplex_t *ap,
                                const armpl_int_t *ipiv, float anorm,
                                float *rcond, armpl_singlecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cspcon_work*, *LAPACKE_dspcon_work*, *LAPACKE_sspcon_work* and *LAPACKE_zspcon_work*.

4.19.444 LAPACKE_csprfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csprfs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_singlecomplex_t *ap,
                           const armpl_singlecomplex_t *afp,
                           const armpl_int_t *ipiv,
                           const armpl_singlecomplex_t *b, armpl_int_t ldb,
                           armpl_singlecomplex_t *x, armpl_int_t ldx,
                           float *ferr, float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csprfs*, *LAPACKE_dsprfs*, *LAPACKE_ssprfs* and *LAPACKE_zsprfs*. It also exists with a native Fortran interface as *csprfs*.

4.19.445 LAPACKE_csprfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csprfs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const armpl_singlecomplex_t *ap,
                                const armpl_singlecomplex_t *afp,
                                const armpl_int_t *ipiv,
                                const armpl_singlecomplex_t *b,
                                armpl_int_t ldb, armpl_singlecomplex_t *x,
                                armpl_int_t ldx, float *ferr, float *berr,
                                armpl_singlecomplex_t *work, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csprfs_work*, *LAPACKE_dsprfs_work*, *LAPACKE_ssprfs_work* and *LAPACKE_zsprfs_work*.

4.19.446 LAPACKE_cspsv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cspsv(armpl_int_t matrix_layout, char uplo, armpl_int_t n,
                          armpl_int_t nrhs, armpl_singlecomplex_t *ap,
                          armpl_int_t *ipiv, armpl_singlecomplex_t *b,
                          armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cspsv*, *LAPACKE_dspsv*, *LAPACKE_sspsv* and *LAPACKE_zspsv*. It also exists with a native Fortran interface as *cspsv*.

4.19.447 LAPACKE_cspsv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cspsv_work(armpl_int_t matrix_layout, char uplo,
                               armpl_int_t n, armpl_int_t nrhs,
                               armpl_singlecomplex_t *ap, armpl_int_t *ipiv,
                               armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cspsv_work*, *LAPACKE_dspsv_work*, *LAPACKE_sspsv_work* and *LAPACKE_zspsv_work*.

4.19.448 LAPACKE_cspsvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cspsvx(armpl_int_t matrix_layout, char fact, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_singlecomplex_t *ap,
                           armpl_singlecomplex_t *afp, armpl_int_t *ipiv,
                           const armpl_singlecomplex_t *b, armpl_int_t ldb,
                           armpl_singlecomplex_t *x, armpl_int_t ldx,
                           float *rcond, float *ferr, float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cspsvx*, *LAPACKE_dspsvx*, *LAPACKE_sspsvx* and *LAPACKE_zspsvx*. It also exists with a native Fortran interface as *cspsvx*.

4.19.449 LAPACKE_cspsvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cspsvx_work(armpl_int_t matrix_layout, char fact,
                                char uplo, armpl_int_t n, armpl_int_t nrhs,
                                const armpl_singlecomplex_t *ap,
                                armpl_singlecomplex_t *afp, armpl_int_t *ipiv,
                                const armpl_singlecomplex_t *b,
                                armpl_int_t ldb, armpl_singlecomplex_t *x,
                                armpl_int_t ldx, float *rcond, float *ferr,
                                float *berr, armpl_singlecomplex_t *work,
                                float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cspsvx_work*, *LAPACKE_dspsvx_work*, *LAPACKE_sspsvx_work* and *LAPACKE_zspsvx_work*.

4.19.450 LAPACKE_csptrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csptrf(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_singlecomplex_t *ap,
                           armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csptrf*, *LAPACKE_dsptrf*, *LAPACKE_ssptrf* and *LAPACKE_zsptrf*. It also exists with a native Fortran interface as *csptrf*.

4.19.451 LAPACKE_csptrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csptrf_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_singlecomplex_t *ap,
                                armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csptrf_work*, *LAPACKE_dsptrf_work*, *LAPACKE_ssptrf_work* and *LAPACKE_zsptrf_work*.

4.19.452 LAPACKE_csptri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csptri(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_singlecomplex_t *ap,
                           const armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csptri*, *LAPACKE_dsptri*, *LAPACKE_ssptri* and *LAPACKE_zsptri*. It also exists with a native Fortran interface as *csptri*.

4.19.453 LAPACKE_csptri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csptri_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_singlecomplex_t *ap,
                                const armpl_int_t *ipiv,
                                armpl_singlecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csptri_work*, *LAPACKE_dsptri_work*, *LAPACKE_ssptri_work* and *LAPACKE_zsptri_work*.

4.19.454 LAPACKE_csptrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csptrs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_singlecomplex_t *ap,
                           const armpl_int_t *ipiv, armpl_singlecomplex_t *b,
                           armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csptrs*, *LAPACKE_dsptrs*, *LAPACKE_ssptrs* and *LAPACKE_zsptrs*. It also exists with a native Fortran interface as *csptrs*.

4.19.455 LAPACKE_csptrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csptrs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const armpl_singlecomplex_t *ap,
                                const armpl_int_t *ipiv,
                                armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csptrs_work*, *LAPACKE_dsptrs_work*, *LAPACKE_ssptrs_work* and *LAPACKE_zsptrs_work*.

4.19.456 LAPACKE_cstedc

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cstedc(armpl_int_t matrix_layout, char compz,
                           armpl_int_t n, float *d, float *e,
                           armpl_singlecomplex_t *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cstedc*, *LAPACKE_dstedc*, *LAPACKE_sstedc* and *LAPACKE_zstedc*. It also exists with a native Fortran interface as *cstedc*.

4.19.457 LAPACKE_cstedc_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cstedc_work(armpl_int_t matrix_layout, char compz,
                                armpl_int_t n, float *d, float *e,
                                armpl_singlecomplex_t *z, armpl_int_t ldz,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork, float *rwork,
                                armpl_int_t lrwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cstedc_work*, *LAPACKE_dstedc_work*, *LAPACKE_sstedc_work* and *LAPACKE_zstedc_work*.

4.19.458 LAPACKE_cstegr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cstegr(armpl_int_t matrix_layout, char jobz, char range,
                           armpl_int_t n, float *d, float *e, float vl,
                           float vu, armpl_int_t il, armpl_int_t iu,
                           float abstol, armpl_int_t *m, float *w,
                           armpl_singlecomplex_t *z, armpl_int_t ldz,
                           armpl_int_t *isuppz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cstegr*, *LAPACKE_dstegr*, *LAPACKE_sstegr* and *LAPACKE_zstegr*. It also exists with a native Fortran interface as *cstegr*.

4.19.459 LAPACKE_cstegr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cstegr_work(armpl_int_t matrix_layout, char jobz,
                                char range, armpl_int_t n, float *d, float *e,
                                float vl, float vu, armpl_int_t il,
                                armpl_int_t iu, float abstol, armpl_int_t *m,
                                float *w, armpl_singlecomplex_t *z,
                                armpl_int_t ldz, armpl_int_t *isuppz,
                                float *work, armpl_int_t lwork,
                                armpl_int_t *iwork, armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cstegr_work*, *LAPACKE_dstegr_work*, *LAPACKE_sstegr_work* and *LAPACKE_zstegr_work*.

4.19.460 LAPACKE_cstein

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cstein(armpl_int_t matrix_layout, armpl_int_t n,
                           const float *d, const float *e, armpl_int_t m,
                           const float *w, const armpl_int_t *iblock,
                           const armpl_int_t *isplit,
                           armpl_singlecomplex_t *z, armpl_int_t ldz,
                           armpl_int_t *ifailv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cstein*, *LAPACKE_dstein*, *LAPACKE_sstein* and *LAPACKE_zstein*. It also exists with a native Fortran interface as *cstein*.

4.19.461 LAPACKE_cstein_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cstein_work(armpl_int_t matrix_layout, armpl_int_t n,
                                const float *d, const float *e, armpl_int_t m,
                                const float *w, const armpl_int_t *iblock,
                                const armpl_int_t *isplit,
                                armpl_singlecomplex_t *z, armpl_int_t ldz,
                                float *work, armpl_int_t *iwork,
                                armpl_int_t *ifailv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cstein_work*, *LAPACKE_dstein_work*, *LAPACKE_sstein_work* and *LAPACKE_zstein_work*.

4.19.462 LAPACKE_cstemr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cstemr(armpl_int_t matrix_layout, char jobz, char range,
                           armpl_int_t n, float *d, float *e, float vl,
                           float vu, armpl_int_t il, armpl_int_t iu,
                           armpl_int_t *m, float *w, armpl_singlecomplex_t *z,
                           armpl_int_t ldz, armpl_int_t nzc,
                           armpl_int_t *isuppz, armpl_int_t *tryrac);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cstemr*, *LAPACKE_dstemr*, *LAPACKE_sstemr* and *LAPACKE_zstemr*. It also exists with a native Fortran interface as *cstemr*.

4.19.463 LAPACKE_cstemr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cstemr_work(armpl_int_t matrix_layout, char jobz,
                                char range, armpl_int_t n, float *d, float *e,
                                float vl, float vu, armpl_int_t il,
                                armpl_int_t iu, armpl_int_t *m, float *w,
                                armpl_singlecomplex_t *z, armpl_int_t ldz,
                                armpl_int_t nzc, armpl_int_t *isuppz,
                                armpl_int_t *tryrac, float *work,
                                armpl_int_t lwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cstemr_work*, *LAPACKE_dstemr_work*, *LAPACKE_sstemr_work* and *LAPACKE_zstemr_work*.

4.19.464 LAPACKE_csteqr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csteqr(armpl_int_t matrix_layout, char compz,
                           armpl_int_t n, float *d, float *e,
                           armpl_singlecomplex_t *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csteqr*, *LAPACKE_dsteqr*, *LAPACKE_ssteqr* and *LAPACKE_zsteqr*. It also exists with a native Fortran interface as *csteqr*.

4.19.465 LAPACKE_csteqr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csteqr_work(armpl_int_t matrix_layout, char compz,
                                armpl_int_t n, float *d, float *e,
                                armpl_singlecomplex_t *z, armpl_int_t ldz,
                                float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csteqr_work*, *LAPACKE_dsteqr_work*, *LAPACKE_ssteqr_work* and *LAPACKE_zsteqr_work*.

4.19.466 LAPACKE_csycon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csycon(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, const armpl_singlecomplex_t *a,
                           armpl_int_t lda, const armpl_int_t *ipiv,
                           float anorm, float *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csycon*, *LAPACKE_dsycon*, *LAPACKE_ssycon* and *LAPACKE_zsycon*. It also exists with a native Fortran interface as *csycon*.

4.19.467 LAPACKE_csycon_3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csycon_3(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, const armpl_singlecomplex_t *a,
                             armpl_int_t lda, const armpl_singlecomplex_t *e,
                             const armpl_int_t *ipiv, float anorm,
                             float *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csycon_3*, *LAPACKE_dsycon_3*, *LAPACKE_ssycon_3* and *LAPACKE_zsycon_3*. It also exists with a native Fortran interface as *csycon_3*.

4.19.468 LAPACKE_csycon_3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csycon_3_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n,
                                  const armpl_singlecomplex_t *a,
                                  armpl_int_t lda,
                                  const armpl_singlecomplex_t *e,
                                  const armpl_int_t *ipiv, float anorm,
                                  float *rcond, armpl_singlecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csycon_3_work*, *LAPACKE_dsycon_3_work*, *LAPACKE_ssycon_3_work* and *LAPACKE_zsycon_3_work*.

4.19.469 LAPACKE_csycon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csycon_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, const armpl_singlecomplex_t *a,
                                armpl_int_t lda, const armpl_int_t *ipiv,
                                float anorm, float *rcond,
                                armpl_singlecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csycon_work*, *LAPACKE_dsycon_work*, *LAPACKE_ssycon_work* and *LAPACKE_zsycon_work*.

4.19.470 LAPACKE_csyconv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csyconv(armpl_int_t matrix_layout, char uplo, char way,
                            armpl_int_t n, armpl_singlecomplex_t *a,
                            armpl_int_t lda, const armpl_int_t *ipiv,
                            armpl_singlecomplex_t *e);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyconv*, *LAPACKE_dsyconv*, *LAPACKE_ssyconv* and *LAPACKE_zsyconv*. It also exists with a native Fortran interface as *csyconv*.

4.19.471 LAPACKE_csyconv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csyconv_work(armpl_int_t matrix_layout, char uplo,
                                 char way, armpl_int_t n,
                                 armpl_singlecomplex_t *a, armpl_int_t lda,
                                 const armpl_int_t *ipiv,
                                 armpl_singlecomplex_t *e);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyconv_work*, *LAPACKE_dsyconv_work*, *LAPACKE_ssyconv_work* and *LAPACKE_zsyconv_work*.

4.19.472 LAPACKE_csyequb

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csyequb(armpl_int_t matrix_layout, char uplo,
                            armpl_int_t n, const armpl_singlecomplex_t *a,
                            armpl_int_t lda, float *s, float *scond,
                            float *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyequb*, *LAPACKE_dsyequb*, *LAPACKE_ssyequb* and *LAPACKE_zsyequb*. It also exists with a native Fortran interface as *csyequb*.

4.19.473 LAPACKE_csyequb_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csyequb_work(armpl_int_t matrix_layout, char uplo,
                                 armpl_int_t n,
                                 const armpl_singlecomplex_t *a,
                                 armpl_int_t lda, float *s, float *scond,
                                 float *amax, armpl_singlecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyequb_work*, *LAPACKE_dsyequb_work*, *LAPACKE_ssyequb_work* and *LAPACKE_zsyequb_work*.

4.19.474 LAPACKE_csyr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csyr(armpl_int_t matrix_layout, char uplo, armpl_int_t n,
                         armpl_singlecomplex_t alpha,
                         const armpl_singlecomplex_t *x, armpl_int_t incx,
                         armpl_singlecomplex_t *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyr* and *LAPACKE_zsyr*. It also exists with a native Fortran interface as *csyr*.

4.19.475 LAPACKE_csyr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csyr_work(armpl_int_t matrix_layout, char uplo,
                              armpl_int_t n, armpl_singlecomplex_t alpha,
                              const armpl_singlecomplex_t *x,
                              armpl_int_t incx, armpl_singlecomplex_t *a,
                              armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyr_work* and *LAPACKE_zsyr_work*.

4.19.476 LAPACKE_csyrfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csyrfs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_singlecomplex_t *a, armpl_int_t lda,
                           const armpl_singlecomplex_t *af, armpl_int_t ldaf,
                           const armpl_int_t *ipiv,
                           const armpl_singlecomplex_t *b, armpl_int_t ldb,
                           armpl_singlecomplex_t *x, armpl_int_t ldx,
                           float *ferr, float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyrfs*, *LAPACKE_dsyrfs*, *LAPACKE_ssyrfs* and *LAPACKE_zsyrfs*. It also exists with a native Fortran interface as *csyrfs*.

4.19.477 LAPACKE_csyrfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csyrfs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
```

(continues on next page)

(continued from previous page)

```
                                const armpl_singlecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_singlecomplex_t *af,
                                armpl_int_t ldaf, const armpl_int_t *ipiv,
                                const armpl_singlecomplex_t *b,
                                armpl_int_t ldb, armpl_singlecomplex_t *x,
                                armpl_int_t ldx, float *ferr, float *berr,
                                armpl_singlecomplex_t *work, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyrfs_work*, *LAPACKE_dsyrfs_work*, *LAPACKE_ssyrfs_work* and *LAPACKE_zsyrfs_work*.

4.19.478 LAPACKE_csyrfsx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csyrfsx(armpl_int_t matrix_layout, char uplo, char equed,
                            armpl_int_t n, armpl_int_t nrhs,
                            const armpl_singlecomplex_t *a, armpl_int_t lda,
                            const armpl_singlecomplex_t *af, armpl_int_t ldaf,
                            const armpl_int_t *ipiv, const float *s,
                            const armpl_singlecomplex_t *b, armpl_int_t ldb,
                            armpl_singlecomplex_t *x, armpl_int_t ldx,
                            float *rcond, float *berr, armpl_int_t n_err_bnds,
                            float *err_bnds_norm, float *err_bnds_comp,
                            armpl_int_t nparams, float *params);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyrfsx*, *LAPACKE_dsyrfsx*, *LAPACKE_ssyrfsx* and *LAPACKE_zsyrfsx*. It also exists with a native Fortran interface as *csyrfsx*.

4.19.479 LAPACKE_csyrfsx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csyrfsx_work(armpl_int_t matrix_layout, char uplo,
                                 char equed, armpl_int_t n, armpl_int_t nrhs,
                                 const armpl_singlecomplex_t *a,
                                 armpl_int_t lda,
                                 const armpl_singlecomplex_t *af,
                                 armpl_int_t ldaf, const armpl_int_t *ipiv,
                                 const float *s,
                                 const armpl_singlecomplex_t *b,
                                 armpl_int_t ldb, armpl_singlecomplex_t *x,
                                 armpl_int_t ldx, float *rcond, float *berr,
                                 armpl_int_t n_err_bnds, float *err_bnds_norm,
                                 float *err_bnds_comp, armpl_int_t nparams,
                                 float *params, armpl_singlecomplex_t *work,
                                 float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyrfsx_work*, *LAPACKE_dsyrfsx_work*, *LAPACKE_ssyrfsx_work* and *LAPACKE_zsyrfsx_work*.

4.19.480 LAPACKE_csysv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csysv(armpl_int_t matrix_layout, char uplo, armpl_int_t n,
                          armpl_int_t nrhs, armpl_singlecomplex_t *a,
                          armpl_int_t lda, armpl_int_t *ipiv,
                          armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysv*, *LAPACKE_dsysv*, *LAPACKE_ssysv* and *LAPACKE_zsysv*. It also exists with a native Fortran interface as *csysv*.

4.19.481 LAPACKE_csysv_aa

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csysv_aa(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, armpl_int_t nrhs,
                             armpl_singlecomplex_t *a, armpl_int_t lda,
                             armpl_int_t *ipiv, armpl_singlecomplex_t *b,
                             armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysv_aa*, *LAPACKE_dsysv_aa*, *LAPACKE_ssysv_aa* and *LAPACKE_zsysv_aa*. It also exists with a native Fortran interface as *csysv_aa*.

4.19.482 LAPACKE_csysv_aa_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csysv_aa_2stage(armpl_int_t matrix_layout, char uplo,
                                    armpl_int_t n, armpl_int_t nrhs,
                                    armpl_singlecomplex_t *a, armpl_int_t lda,
                                    armpl_singlecomplex_t *tb,
```

(continues on next page)

(continued from previous page)

```
                                    armpl_int_t ltb, armpl_int_t *ipiv,
                                    armpl_int_t *ipiv2,
                                    armpl_singlecomplex_t *b,
                                    armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysv_aa_2stage*, *LAPACKE_dsysv_aa_2stage*, *LAPACKE_ssysv_aa_2stage* and *LAPACKE_zsysv_aa_2stage*. It also exists with a native Fortran interface as *csysv_aa_2stage*.

4.19.483 LAPACKE_csysv_aa_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csysv_aa_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, armpl_int_t nrhs,
                                  armpl_singlecomplex_t *a, armpl_int_t lda,
                                  armpl_int_t *ipiv, armpl_singlecomplex_t *b,
                                  armpl_int_t ldb,
                                  armpl_singlecomplex_t *work,
                                  armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysv_aa_work*, *LAPACKE_dsysv_aa_work*, *LAPACKE_ssysv_aa_work* and *LAPACKE_zsysv_aa_work*.

4.19.484 LAPACKE_csysv_rk

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csysv_rk(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, armpl_int_t nrhs,
                             armpl_singlecomplex_t *a, armpl_int_t lda,
                             armpl_singlecomplex_t *e, armpl_int_t *ipiv,
                             armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysv_rk*, *LAPACKE_dsysv_rk*, *LAPACKE_ssysv_rk* and *LAPACKE_zsysv_rk*. It also exists with a native Fortran interface as *csysv_rk*.

4.19.485 LAPACKE_csysv_rk_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csysv_rk_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, armpl_int_t nrhs,
                                  armpl_singlecomplex_t *a, armpl_int_t lda,
                                  armpl_singlecomplex_t *e, armpl_int_t *ipiv,
                                  armpl_singlecomplex_t *b, armpl_int_t ldb,
                                  armpl_singlecomplex_t *work,
                                  armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysv_rk_work*, *LAPACKE_dsysv_rk_work*, *LAPACKE_ssysv_rk_work* and *LAPACKE_zsysv_rk_work*.

4.19.486 LAPACKE_csysv_rook

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csysv_rook(armpl_int_t matrix_layout, char uplo,
                               armpl_int_t n, armpl_int_t nrhs,
                               armpl_singlecomplex_t *a, armpl_int_t lda,
                               armpl_int_t *ipiv, armpl_singlecomplex_t *b,
                               armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysv_rook*, *LAPACKE_dsysv_rook*, *LAPACKE_ssysv_rook* and *LAPACKE_zsysv_rook*. It also exists with a native Fortran interface as *csysv_rook*.

4.19.487 LAPACKE_csysv_rook_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csysv_rook_work(armpl_int_t matrix_layout, char uplo,
                                    armpl_int_t n, armpl_int_t nrhs,
                                    armpl_singlecomplex_t *a, armpl_int_t lda,
                                    armpl_int_t *ipiv,
                                    armpl_singlecomplex_t *b, armpl_int_t ldb,
                                    armpl_singlecomplex_t *work,
                                    armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysv_rook_work*, *LAPACKE_dsysv_rook_work*, *LAPACKE_ssysv_rook_work* and *LAPACKE_zsysv_rook_work*.

4.19.488 LAPACKE_csysv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csysv_work(armpl_int_t matrix_layout, char uplo,
                               armpl_int_t n, armpl_int_t nrhs,
                               armpl_singlecomplex_t *a, armpl_int_t lda,
                               armpl_int_t *ipiv, armpl_singlecomplex_t *b,
                               armpl_int_t ldb, armpl_singlecomplex_t *work,
                               armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysv_work*, *LAPACKE_dsysv_work*, *LAPACKE_ssysv_work* and *LAPACKE_zsysv_work*.

4.19.489 LAPACKE_csysvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csysvx(armpl_int_t matrix_layout, char fact, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_singlecomplex_t *a, armpl_int_t lda,
                           armpl_singlecomplex_t *af, armpl_int_t ldaf,
                           armpl_int_t *ipiv, const armpl_singlecomplex_t *b,
                           armpl_int_t ldb, armpl_singlecomplex_t *x,
                           armpl_int_t ldx, float *rcond, float *ferr,
                           float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysvx*, *LAPACKE_dsysvx*, *LAPACKE_ssysvx* and *LAPACKE_zsysvx*. It also exists with a native Fortran interface as *csysvx*.

4.19.490 LAPACKE_csysvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csysvx_work(armpl_int_t matrix_layout, char fact,
                                char uplo, armpl_int_t n, armpl_int_t nrhs,
                                const armpl_singlecomplex_t *a,
                                armpl_int_t lda, armpl_singlecomplex_t *af,
                                armpl_int_t ldaf, armpl_int_t *ipiv,
                                const armpl_singlecomplex_t *b,
                                armpl_int_t ldb, armpl_singlecomplex_t *x,
                                armpl_int_t ldx, float *rcond, float *ferr,
                                float *berr, armpl_singlecomplex_t *work,
                                armpl_int_t lwork, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysvx_work*, *LAPACKE_dsysvx_work*, *LAPACKE_ssysvx_work* and *LAPACKE_zsysvx_work*.

4.19.491 LAPACKE_csysvxx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csysvxx(armpl_int_t matrix_layout, char fact, char uplo,
                            armpl_int_t n, armpl_int_t nrhs,
                            armpl_singlecomplex_t *a, armpl_int_t lda,
                            armpl_singlecomplex_t *af, armpl_int_t ldaf,
                            armpl_int_t *ipiv, char *equed, float *s,
                            armpl_singlecomplex_t *b, armpl_int_t ldb,
                            armpl_singlecomplex_t *x, armpl_int_t ldx,
                            float *rcond, float *rpvgrw, float *berr,
                            armpl_int_t n_err_bnds, float *err_bnds_norm,
                            float *err_bnds_comp, armpl_int_t nparams,
                            float *params);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysvxx*, *LAPACKE_dsysvxx*, *LAPACKE_ssysvxx* and *LAPACKE_zsysvxx*. It also exists with a native Fortran interface as *csysvxx*.

4.19.492 LAPACKE_csysvxx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csysvxx_work(armpl_int_t matrix_layout, char fact,
                                 char uplo, armpl_int_t n, armpl_int_t nrhs,
                                 armpl_singlecomplex_t *a, armpl_int_t lda,
                                 armpl_singlecomplex_t *af, armpl_int_t ldaf,
                                 armpl_int_t *ipiv, char *equed, float *s,
                                 armpl_singlecomplex_t *b, armpl_int_t ldb,
                                 armpl_singlecomplex_t *x, armpl_int_t ldx,
                                 float *rcond, float *rpvgrw, float *berr,
                                 armpl_int_t n_err_bnds, float *err_bnds_norm,
                                 float *err_bnds_comp, armpl_int_t nparams,
                                 float *params, armpl_singlecomplex_t *work,
                                 float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysvxx_work*, *LAPACKE_dsysvxx_work*, *LAPACKE_ssysvxx_work* and *LAPACKE_zsysvxx_work*.

4.19.493 LAPACKE_csyswapr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csyswapr(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, armpl_singlecomplex_t *a,
                             armpl_int_t lda, armpl_int_t i1,
                             armpl_int_t i2);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyswapr*, *LAPACKE_dsyswapr*, *LAPACKE_ssyswapr* and *LAPACKE_zsyswapr*. It also exists with a native Fortran interface as *csyswapr*.

4.19.494 LAPACKE_csyswapr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csyswapr_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, armpl_singlecomplex_t *a,
                                  armpl_int_t lda, armpl_int_t i1,
                                  armpl_int_t i2);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyswapr_work*, *LAPACKE_dsyswapr_work*, *LAPACKE_ssyswapr_work* and *LAPACKE_zsyswapr_work*.

4.19.495 LAPACKE_csytrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csytrf(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_singlecomplex_t *a,
                           armpl_int_t lda, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrf*, *LAPACKE_dsytrf*, *LAPACKE_ssytrf* and *LAPACKE_zsytrf*. It also exists with a native Fortran interface as *csytrf*.

4.19.496 LAPACKE_csytrf_aa

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csytrf_aa(armpl_int_t matrix_layout, char uplo,
                              armpl_int_t n, armpl_singlecomplex_t *a,
                              armpl_int_t lda, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrf_aa*, *LAPACKE_dsytrf_aa*, *LAPACKE_ssytrf_aa* and *LAPACKE_zsytrf_aa*. It also exists with a native Fortran interface as *csytrf_aa*.

4.19.497 LAPACKE_csytrf_aa_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csytrf_aa_2stage(armpl_int_t matrix_layout, char uplo,
                                     armpl_int_t n, armpl_singlecomplex_t *a,
                                     armpl_int_t lda,
                                     armpl_singlecomplex_t *tb,
                                     armpl_int_t ltb, armpl_int_t *ipiv,
                                     armpl_int_t *ipiv2);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrf_aa_2stage*, *LAPACKE_dsytrf_aa_2stage*, *LAPACKE_ssytrf_aa_2stage* and *LAPACKE_zsytrf_aa_2stage*. It also exists with a native Fortran interface as *csytrf_aa_2stage*.

4.19.498 LAPACKE_csytrf_aa_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csytrf_aa_work(armpl_int_t matrix_layout, char uplo,
                                   armpl_int_t n, armpl_singlecomplex_t *a,
                                   armpl_int_t lda, armpl_int_t *ipiv,
                                   armpl_singlecomplex_t *work,
                                   armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrf_aa_work*, *LAPACKE_dsytrf_aa_work*, *LAPACKE_ssytrf_aa_work* and *LAPACKE_zsytrf_aa_work*.

4.19.499 LAPACKE_csytrf_rk

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csytrf_rk(armpl_int_t matrix_layout, char uplo,
                              armpl_int_t n, armpl_singlecomplex_t *a,
                              armpl_int_t lda, armpl_singlecomplex_t *e,
                              armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrf_rk*, *LAPACKE_dsytrf_rk*, *LAPACKE_ssytrf_rk* and *LAPACKE_zsytrf_rk*. It also exists with a native Fortran interface as *csytrf_rk*.

4.19.500 LAPACKE_csytrf_rk_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csytrf_rk_work(armpl_int_t matrix_layout, char uplo,
                                   armpl_int_t n, armpl_singlecomplex_t *a,
                                   armpl_int_t lda, armpl_singlecomplex_t *e,
                                   armpl_int_t *ipiv,
                                   armpl_singlecomplex_t *work,
                                   armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrf_rk_work*, *LAPACKE_dsytrf_rk_work*, *LAPACKE_ssytrf_rk_work* and *LAPACKE_zsytrf_rk_work*.

4.19.501 LAPACKE_csytrf_rook

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csytrf_rook(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_singlecomplex_t *a,
                                armpl_int_t lda, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrf_rook*, *LAPACKE_dsytrf_rook*, *LAPACKE_ssytrf_rook* and *LAPACKE_zsytrf_rook*. It also exists with a native Fortran interface as *csytrf_rook*.

4.19.502 LAPACKE_csytrf_rook_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csytrf_rook_work(armpl_int_t matrix_layout, char uplo,
                                     armpl_int_t n, armpl_singlecomplex_t *a,
                                     armpl_int_t lda, armpl_int_t *ipiv,
                                     armpl_singlecomplex_t *work,
                                     armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrf_rook_work*, *LAPACKE_dsytrf_rook_work*, *LAPACKE_ssytrf_rook_work* and *LAPACKE_zsytrf_rook_work*.

4.19.503 LAPACKE_csytrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csytrf_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_singlecomplex_t *a,
                                armpl_int_t lda, armpl_int_t *ipiv,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrf_work*, *LAPACKE_dsytrf_work*, *LAPACKE_ssytrf_work* and *LAPACKE_zsytrf_work*.

4.19.504 LAPACKE_csytri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csytri(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_singlecomplex_t *a,
                           armpl_int_t lda, const armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytri*, *LAPACKE_dsytri*, *LAPACKE_ssytri* and *LAPACKE_zsytri*. It also exists with a native Fortran interface as *csytri*.

4.19.505 LAPACKE_csytri2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csytri2(armpl_int_t matrix_layout, char uplo,
                            armpl_int_t n, armpl_singlecomplex_t *a,
                            armpl_int_t lda, const armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytri2*, *LAPACKE_dsytri2*, *LAPACKE_ssytri2* and *LAPACKE_zsytri2*. It also exists with a native Fortran interface as *csytri2*.

4.19.506 LAPACKE_csytri2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csytri2_work(armpl_int_t matrix_layout, char uplo,
                                 armpl_int_t n, armpl_singlecomplex_t *a,
                                 armpl_int_t lda, const armpl_int_t *ipiv,
                                 armpl_singlecomplex_t *work,
                                 armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytri2_work*, *LAPACKE_dsytri2_work*, *LAPACKE_ssytri2_work* and *LAPACKE_zsytri2_work*.

4.19.507 LAPACKE_csytri2x

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csytri2x(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, armpl_singlecomplex_t *a,
                             armpl_int_t lda, const armpl_int_t *ipiv,
                             armpl_int_t nb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytri2x*, *LAPACKE_dsytri2x*, *LAPACKE_ssytri2x* and *LAPACKE_zsytri2x*. It also exists with a native Fortran interface as *csytri2x*.

4.19.508 LAPACKE_csytri2x_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csytri2x_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, armpl_singlecomplex_t *a,
                                  armpl_int_t lda, const armpl_int_t *ipiv,
                                  armpl_singlecomplex_t *work,
                                  armpl_int_t nb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytri2x_work*, *LAPACKE_dsytri2x_work*, *LAPACKE_ssytri2x_work* and *LAPACKE_zsytri2x_work*.

4.19.509 LAPACKE_csytri_3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csytri_3(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, armpl_singlecomplex_t *a,
                             armpl_int_t lda, const armpl_singlecomplex_t *e,
                             const armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytri_3*, *LAPACKE_dsytri_3*, *LAPACKE_ssytri_3* and *LAPACKE_zsytri_3*. It also exists with a native Fortran interface as *csytri_3*.

4.19.510 LAPACKE_csytri_3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csytri_3_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, armpl_singlecomplex_t *a,
                                  armpl_int_t lda,
                                  const armpl_singlecomplex_t *e,
                                  const armpl_int_t *ipiv,
```

(continues on next page)

(continued from previous page)

```
                                    armpl_singlecomplex_t *work,
                                    armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytri_3_work*, *LAPACKE_dsytri_3_work*, *LAPACKE_ssytri_3_work* and *LAPACKE_zsytri_3_work*.

4.19.511 LAPACKE_csytri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csytri_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_singlecomplex_t *a,
                                armpl_int_t lda, const armpl_int_t *ipiv,
                                armpl_singlecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.512 LAPACKE_csytrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csytrs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_singlecomplex_t *a, armpl_int_t lda,
```

(continues on next page)

(continued from previous page)

```
                            const armpl_int_t *ipiv, armpl_singlecomplex_t *b,
                            armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs*, *LAPACKE_dsytrs*, *LAPACKE_ssytrs* and *LAPACKE_zsytrs*. It also exists with a native Fortran interface as *csytrs*.

4.19.513 LAPACKE_csytrs2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csytrs2(armpl_int_t matrix_layout, char uplo,
                            armpl_int_t n, armpl_int_t nrhs,
                            const armpl_singlecomplex_t *a, armpl_int_t lda,
                            const armpl_int_t *ipiv, armpl_singlecomplex_t *b,
                            armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.514 LAPACKE_csytrs2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csytrs2_work(armpl_int_t matrix_layout, char uplo,
                                 armpl_int_t n, armpl_int_t nrhs,
```

(continues on next page)

(continued from previous page)

```
                                const armpl_singlecomplex_t *a,
                                armpl_int_t lda, const armpl_int_t *ipiv,
                                armpl_singlecomplex_t *b, armpl_int_t ldb,
                                armpl_singlecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs2_work*, *LAPACKE_dsytrs2_work*, *LAPACKE_ssytrs2_work* and *LAPACKE_zsytrs2_work*.

4.19.515 LAPACKE_csytrs_3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csytrs_3(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, armpl_int_t nrhs,
                             const armpl_singlecomplex_t *a, armpl_int_t lda,
                             const armpl_singlecomplex_t *e,
                             const armpl_int_t *ipiv,
                             armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs_3*, *LAPACKE_dsytrs_3*, *LAPACKE_ssytrs_3* and *LAPACKE_zsytrs_3*. It also exists with a native Fortran interface as *csytrs_3*.

4.19.516 LAPACKE_csytrs_3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csytrs_3_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, armpl_int_t nrhs,
                                  const armpl_singlecomplex_t *a,
                                  armpl_int_t lda,
                                  const armpl_singlecomplex_t *e,
                                  const armpl_int_t *ipiv,
                                  armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs_3_work*, *LAPACKE_dsytrs_3_work*, *LAPACKE_ssytrs_3_work* and *LAPACKE_zsytrs_3_work*.

4.19.517 LAPACKE_csytrs_aa

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csytrs_aa(armpl_int_t matrix_layout, char uplo,
                              armpl_int_t n, armpl_int_t nrhs,
                              const armpl_singlecomplex_t *a, armpl_int_t lda,
                              const armpl_int_t *ipiv,
                              armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs_aa*, *LAPACKE_dsytrs_aa*, *LAPACKE_ssytrs_aa* and *LAPACKE_zsytrs_aa*. It also exists with a native Fortran interface as *csytrs_aa*.

4.19.518 LAPACKE_csytrs_aa_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csytrs_aa_2stage(armpl_int_t matrix_layout, char uplo,
                                     armpl_int_t n, armpl_int_t nrhs,
                                     armpl_singlecomplex_t *a,
                                     armpl_int_t lda,
                                     armpl_singlecomplex_t *tb,
                                     armpl_int_t ltb, armpl_int_t *ipiv,
                                     armpl_int_t *ipiv2,
                                     armpl_singlecomplex_t *b,
                                     armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs_aa_2stage*, *LAPACKE_dsytrs_aa_2stage*, *LAPACKE_ssytrs_aa_2stage* and *LAPACKE_zsytrs_aa_2stage*. It also exists with a native Fortran interface as *csytrs_aa_2stage*.

4.19.519 LAPACKE_csytrs_aa_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csytrs_aa_work(armpl_int_t matrix_layout, char uplo,
                                   armpl_int_t n, armpl_int_t nrhs,
                                   const armpl_singlecomplex_t *a,
                                   armpl_int_t lda, const armpl_int_t *ipiv,
                                   armpl_singlecomplex_t *b, armpl_int_t ldb,
                                   armpl_singlecomplex_t *work,
                                   armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs_aa_work*, *LAPACKE_dsytrs_aa_work*, *LAPACKE_ssytrs_aa_work* and *LAPACKE_zsytrs_aa_work*.

4.19.520 LAPACKE_csytrs_rook

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csytrs_rook(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const armpl_singlecomplex_t *a,
                                armpl_int_t lda, const armpl_int_t *ipiv,
                                armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs_rook*, *LAPACKE_dsytrs_rook*, *LAPACKE_ssytrs_rook* and *LAPACKE_zsytrs_rook*. It also exists with a native Fortran interface as *csytrs_rook*.

4.19.521 LAPACKE_csytrs_rook_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csytrs_rook_work(armpl_int_t matrix_layout, char uplo,
                                     armpl_int_t n, armpl_int_t nrhs,
                                     const armpl_singlecomplex_t *a,
                                     armpl_int_t lda, const armpl_int_t *ipiv,
```

(continues on next page)

(continued from previous page)

```
                                        armpl_singlecomplex_t *b,
                                        armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs_rook_work*, *LAPACKE_dsytrs_rook_work*, *LAPACKE_ssytrs_rook_work* and *LAPACKE_zsytrs_rook_work*.

4.19.522 LAPACKE_csytrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_csytrs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const armpl_singlecomplex_t *a,
                                armpl_int_t lda, const armpl_int_t *ipiv,
                                armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs_work*, *LAPACKE_dsytrs_work*, *LAPACKE_ssytrs_work* and *LAPACKE_zsytrs_work*.

4.19.523 LAPACKE_ctbcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctbcon(armpl_int_t matrix_layout, char norm, char uplo,
                           char diag, armpl_int_t n, armpl_int_t kd,
                           const armpl_singlecomplex_t *ab, armpl_int_t ldab,
                           float *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctbcon*, *LAPACKE_dtbcon*, *LAPACKE_stbcon* and *LAPACKE_ztbcon*. It also exists with a native Fortran interface as *ctbcon*.

4.19.524 LAPACKE_ctbcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctbcon_work(armpl_int_t matrix_layout, char norm,
                                char uplo, char diag, armpl_int_t n,
                                armpl_int_t kd,
                                const armpl_singlecomplex_t *ab,
                                armpl_int_t ldab, float *rcond,
                                armpl_singlecomplex_t *work, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctbcon_work*, *LAPACKE_dtbcon_work*, *LAPACKE_stbcon_work* and *LAPACKE_ztbcon_work*.

4.19.525 LAPACKE_ctbrfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctbrfs(armpl_int_t matrix_layout, char uplo, char trans,
                           char diag, armpl_int_t n, armpl_int_t kd,
                           armpl_int_t nrhs, const armpl_singlecomplex_t *ab,
                           armpl_int_t ldab, const armpl_singlecomplex_t *b,
                           armpl_int_t ldb, const armpl_singlecomplex_t *x,
                           armpl_int_t ldx, float *ferr, float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctbrfs*, *LAPACKE_dtbrfs*, *LAPACKE_stbrfs* and *LAPACKE_ztbrfs*. It also exists with a native Fortran interface as *ctbrfs*.

4.19.526 LAPACKE_ctbrfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctbrfs_work(armpl_int_t matrix_layout, char uplo,
                                char trans, char diag, armpl_int_t n,
                                armpl_int_t kd, armpl_int_t nrhs,
                                const armpl_singlecomplex_t *ab,
                                armpl_int_t ldab,
                                const armpl_singlecomplex_t *b,
                                armpl_int_t ldb,
                                const armpl_singlecomplex_t *x,
                                armpl_int_t ldx, float *ferr, float *berr,
                                armpl_singlecomplex_t *work, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctbrfs_work*, *LAPACKE_dtbrfs_work*, *LAPACKE_stbrfs_work* and *LAPACKE_ztbrfs_work*.

4.19.527 LAPACKE_ctbtrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctbtrs(armpl_int_t matrix_layout, char uplo, char trans,
                           char diag, armpl_int_t n, armpl_int_t kd,
                           armpl_int_t nrhs, const armpl_singlecomplex_t *ab,
                           armpl_int_t ldab, armpl_singlecomplex_t *b,
                           armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctbtrs*, *LAPACKE_dtbtrs*, *LAPACKE_stbtrs* and *LAPACKE_ztbtrs*. It also exists with a native Fortran interface as *ctbtrs*.

4.19.528 LAPACKE_ctbtrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctbtrs_work(armpl_int_t matrix_layout, char uplo,
                                char trans, char diag, armpl_int_t n,
                                armpl_int_t kd, armpl_int_t nrhs,
                                const armpl_singlecomplex_t *ab,
                                armpl_int_t ldab, armpl_singlecomplex_t *b,
                                armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctbtrs_work*, *LAPACKE_dtbtrs_work*, *LAPACKE_stbtrs_work* and *LAPACKE_ztbtrs_work*.

4.19.529 LAPACKE_ctfsm

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctfsm(armpl_int_t matrix_layout, char transr, char side,
                          char uplo, char trans, char diag, armpl_int_t m,
                          armpl_int_t n, armpl_singlecomplex_t alpha,
                          const armpl_singlecomplex_t *a,
                          armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctfsm*, *LAPACKE_dtfsm*, *LAPACKE_stfsm* and *LAPACKE_ztfsm*. It also exists with a native Fortran interface as *ctfsm*.

4.19.530 LAPACKE_ctfsm_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctfsm_work(armpl_int_t matrix_layout, char transr,
                               char side, char uplo, char trans, char diag,
                               armpl_int_t m, armpl_int_t n,
                               armpl_singlecomplex_t alpha,
                               const armpl_singlecomplex_t *a,
                               armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctfsm_work*, *LAPACKE_dtfsm_work*, *LAPACKE_stfsm_work* and *LAPACKE_ztfsm_work*.

4.19.531 LAPACKE_ctftri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctftri(armpl_int_t matrix_layout, char transr, char uplo,
                           char diag, armpl_int_t n,
                           armpl_singlecomplex_t *a);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctftri*, *LAPACKE_dtftri*, *LAPACKE_stftri* and *LAPACKE_ztftri*. It also exists with a native Fortran interface as *ctftri*.

4.19.532 LAPACKE_ctftri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctftri_work(armpl_int_t matrix_layout, char transr,
                                char uplo, char diag, armpl_int_t n,
                                armpl_singlecomplex_t *a);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctftri_work*, *LAPACKE_dtftri_work*, *LAPACKE_stftri_work* and *LAPACKE_ztftri_work*.

4.19.533 LAPACKE_ctfttp

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctfttp(armpl_int_t matrix_layout, char transr, char uplo,
                           armpl_int_t n, const armpl_singlecomplex_t *arf,
                           armpl_singlecomplex_t *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctfttp*, *LAPACKE_dtfttp*, *LAPACKE_stfttp* and *LAPACKE_ztfttp*. It also exists with a native Fortran interface as *ctfttp*.

4.19.534 LAPACKE_ctfttp_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctfttp_work(armpl_int_t matrix_layout, char transr,
                                char uplo, armpl_int_t n,
                                const armpl_singlecomplex_t *arf,
                                armpl_singlecomplex_t *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctfttp_work*, *LAPACKE_dtfttp_work*, *LAPACKE_stfttp_work* and *LAPACKE_ztfttp_work*.

4.19.535 LAPACKE_ctfttr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctfttr(armpl_int_t matrix_layout, char transr, char uplo,
                           armpl_int_t n, const armpl_singlecomplex_t *arf,
                           armpl_singlecomplex_t *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctfttr*, *LAPACKE_dtfttr*, *LAPACKE_stfttr* and *LAPACKE_ztfttr*. It also exists with a native Fortran interface as *ctfttr*.

4.19.536 LAPACKE_ctfttr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctfttr_work(armpl_int_t matrix_layout, char transr,
                                char uplo, armpl_int_t n,
                                const armpl_singlecomplex_t *arf,
                                armpl_singlecomplex_t *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctfttr_work*, *LAPACKE_dtfttr_work*, *LAPACKE_stfttr_work* and *LAPACKE_ztfttr_work*.

4.19.537 LAPACKE_ctgevc

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctgevc(armpl_int_t matrix_layout, char side, char howmny,
                           const armpl_int_t *select, armpl_int_t n,
                           const armpl_singlecomplex_t *s, armpl_int_t lds,
                           const armpl_singlecomplex_t *p, armpl_int_t ldp,
                           armpl_singlecomplex_t *vl, armpl_int_t ldvl,
                           armpl_singlecomplex_t *vr, armpl_int_t ldvr,
                           armpl_int_t mm, armpl_int_t *m);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgevc*, *LAPACKE_dtgevc*, *LAPACKE_stgevc* and *LAPACKE_ztgevc*. It also exists with a native Fortran interface as *ctgevc*.

4.19.538 LAPACKE_ctgevc_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctgevc_work(armpl_int_t matrix_layout, char side,
                                char howmny, const armpl_int_t *select,
```

(continues on next page)

(continued from previous page)

```
                                armpl_int_t n, const armpl_singlecomplex_t *s,
                                armpl_int_t lds,
                                const armpl_singlecomplex_t *p,
                                armpl_int_t ldp, armpl_singlecomplex_t *vl,
                                armpl_int_t ldvl, armpl_singlecomplex_t *vr,
                                armpl_int_t ldvr, armpl_int_t mm,
                                armpl_int_t *m, armpl_singlecomplex_t *work,
                                float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgevc_work*, *LAPACKE_dtgevc_work*, *LAPACKE_stgevc_work* and *LAPACKE_ztgevc_work*.

4.19.539 LAPACKE_ctgexc

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctgexc(armpl_int_t matrix_layout, armpl_int_t wantq,
                           armpl_int_t wantz, armpl_int_t n,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           armpl_singlecomplex_t *b, armpl_int_t ldb,
                           armpl_singlecomplex_t *q, armpl_int_t ldq,
                           armpl_singlecomplex_t *z, armpl_int_t ldz,
                           armpl_int_t ifst, armpl_int_t ilst);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgexc*, *LAPACKE_dtgexc*, *LAPACKE_stgexc* and *LAPACKE_ztgexc*. It also exists with a native Fortran interface as *ctgexc*.

4.19.540 LAPACKE_ctgexc_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctgexc_work(armpl_int_t matrix_layout, armpl_int_t wantq,
                                armpl_int_t wantz, armpl_int_t n,
                                armpl_singlecomplex_t *a, armpl_int_t lda,
                                armpl_singlecomplex_t *b, armpl_int_t ldb,
                                armpl_singlecomplex_t *q, armpl_int_t ldq,
                                armpl_singlecomplex_t *z, armpl_int_t ldz,
                                armpl_int_t ifst, armpl_int_t ilst);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgexc_work*, *LAPACKE_dtgexc_work*, *LAPACKE_stgexc_work* and *LAPACKE_ztgexc_work*.

4.19.541 LAPACKE_ctgsen

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctgsen(armpl_int_t matrix_layout, armpl_int_t ijob,
                           armpl_int_t wantq, armpl_int_t wantz,
                           const armpl_int_t *select, armpl_int_t n,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           armpl_singlecomplex_t *b, armpl_int_t ldb,
                           armpl_singlecomplex_t *alpha,
                           armpl_singlecomplex_t *beta,
                           armpl_singlecomplex_t *q, armpl_int_t ldq,
                           armpl_singlecomplex_t *z, armpl_int_t ldz,
                           armpl_int_t *m, float *pl, float *pr, float *dif);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgsen*, *LAPACKE_dtgsen*, *LAPACKE_stgsen* and *LAPACKE_ztgsen*. It also exists with a native Fortran interface as *ctgsen*.

4.19.542 LAPACKE_ctgsen_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctgsen_work(armpl_int_t matrix_layout, armpl_int_t ijob,
                                armpl_int_t wantq, armpl_int_t wantz,
                                const armpl_int_t *select, armpl_int_t n,
                                armpl_singlecomplex_t *a, armpl_int_t lda,
                                armpl_singlecomplex_t *b, armpl_int_t ldb,
                                armpl_singlecomplex_t *alpha,
                                armpl_singlecomplex_t *beta,
                                armpl_singlecomplex_t *q, armpl_int_t ldq,
                                armpl_singlecomplex_t *z, armpl_int_t ldz,
                                armpl_int_t *m, float *pl, float *pr,
                                float *dif, armpl_singlecomplex_t *work,
                                armpl_int_t lwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgsen_work*, *LAPACKE_dtgsen_work*, *LAPACKE_stgsen_work* and *LAPACKE_ztgsen_work*.

4.19.543 LAPACKE_ctgsja

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctgsja(armpl_int_t matrix_layout, char jobu, char jobv,
                           char jobq, armpl_int_t m, armpl_int_t p,
                           armpl_int_t n, armpl_int_t k, armpl_int_t l,
```

(continues on next page)

(continued from previous page)

```
                                armpl_singlecomplex_t *a, armpl_int_t lda,
                                armpl_singlecomplex_t *b, armpl_int_t ldb,
                                float tola, float tolb, float *alpha, float *beta,
                                armpl_singlecomplex_t *u, armpl_int_t ldu,
                                armpl_singlecomplex_t *v, armpl_int_t ldv,
                                armpl_singlecomplex_t *q, armpl_int_t ldq,
                                armpl_int_t *ncycle);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgsja*, *LAPACKE_dtgsja*, *LAPACKE_stgsja* and *LAPACKE_ztgsja*. It also exists with a native Fortran interface as *ctgsja*.

4.19.544 LAPACKE_ctgsja_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctgsja_work(armpl_int_t matrix_layout, char jobu,
                                char jobv, char jobq, armpl_int_t m,
                                armpl_int_t p, armpl_int_t n, armpl_int_t k,
                                armpl_int_t l, armpl_singlecomplex_t *a,
                                armpl_int_t lda, armpl_singlecomplex_t *b,
                                armpl_int_t ldb, float tola, float tolb,
                                float *alpha, float *beta,
                                armpl_singlecomplex_t *u, armpl_int_t ldu,
                                armpl_singlecomplex_t *v, armpl_int_t ldv,
                                armpl_singlecomplex_t *q, armpl_int_t ldq,
                                armpl_singlecomplex_t *work,
                                armpl_int_t *ncycle);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgsja_work*, *LAPACKE_dtgsja_work*, *LAPACKE_stgsja_work* and *LAPACKE_ztgsja_work*.

4.19.545 LAPACKE_ctgsna

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctgsna(armpl_int_t matrix_layout, char job, char howmny,
                           const armpl_int_t *select, armpl_int_t n,
                           const armpl_singlecomplex_t *a, armpl_int_t lda,
                           const armpl_singlecomplex_t *b, armpl_int_t ldb,
                           const armpl_singlecomplex_t *vl, armpl_int_t ldvl,
                           const armpl_singlecomplex_t *vr, armpl_int_t ldvr,
                           float *s, float *dif, armpl_int_t mm,
                           armpl_int_t *m);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgsna*, *LAPACKE_dtgsna*, *LAPACKE_stgsna* and *LAPACKE_ztgsna*. It also exists with a native Fortran interface as *ctgsna*.

4.19.546 LAPACKE_ctgsna_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctgsna_work(armpl_int_t matrix_layout, char job,
                                char howmny, const armpl_int_t *select,
                                armpl_int_t n, const armpl_singlecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_singlecomplex_t *b,
                                armpl_int_t ldb,
                                const armpl_singlecomplex_t *vl,
                                armpl_int_t ldvl,
                                const armpl_singlecomplex_t *vr,
                                armpl_int_t ldvr, float *s, float *dif,
                                armpl_int_t mm, armpl_int_t *m,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgsna_work*, *LAPACKE_dtgsna_work*, *LAPACKE_stgsna_work* and *LAPACKE_ztgsna_work*.

4.19.547 LAPACKE_ctgsyl

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctgsyl(armpl_int_t matrix_layout, char trans,
                           armpl_int_t ijob, armpl_int_t m, armpl_int_t n,
                           const armpl_singlecomplex_t *a, armpl_int_t lda,
                           const armpl_singlecomplex_t *b, armpl_int_t ldb,
                           armpl_singlecomplex_t *c, armpl_int_t ldc,
                           const armpl_singlecomplex_t *d, armpl_int_t ldd,
                           const armpl_singlecomplex_t *e, armpl_int_t lde,
                           armpl_singlecomplex_t *f, armpl_int_t ldf,
                           float *scale, float *dif);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgsyl*, *LAPACKE_dtgsyl*, *LAPACKE_stgsyl* and *LAPACKE_ztgsyl*. It also exists with a native Fortran interface as *ctgsyl*.

4.19.548 LAPACKE_ctgsyl_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctgsyl_work(armpl_int_t matrix_layout, char trans,
                                armpl_int_t ijob, armpl_int_t m,
                                armpl_int_t n, const armpl_singlecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_singlecomplex_t *b,
                                armpl_int_t ldb, armpl_singlecomplex_t *c,
                                armpl_int_t ldc,
                                const armpl_singlecomplex_t *d,
                                armpl_int_t ldd,
                                const armpl_singlecomplex_t *e,
                                armpl_int_t lde, armpl_singlecomplex_t *f,
                                armpl_int_t ldf, float *scale, float *dif,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgsyl_work*, *LAPACKE_dtgsyl_work*, *LAPACKE_stgsyl_work* and *LAPACKE_ztgsyl_work*.

4.19.549 LAPACKE_ctpcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctpcon(armpl_int_t matrix_layout, char norm, char uplo,
                           char diag, armpl_int_t n,
                           const armpl_singlecomplex_t *ap, float *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpcon*, *LAPACKE_dtpcon*, *LAPACKE_stpcon* and *LAPACKE_ztpcon*. It also exists with a native Fortran interface as *ctpcon*.

4.19.550 LAPACKE_ctpcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctpcon_work(armpl_int_t matrix_layout, char norm,
                                char uplo, char diag, armpl_int_t n,
                                const armpl_singlecomplex_t *ap, float *rcond,
                                armpl_singlecomplex_t *work, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpcon_work*, *LAPACKE_dtpcon_work*, *LAPACKE_stpcon_work* and *LAPACKE_ztpcon_work*.

4.19.551 LAPACKE_ctpmqrt

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctpmqrt(armpl_int_t matrix_layout, char side, char trans,
                            armpl_int_t m, armpl_int_t n, armpl_int_t k,
                            armpl_int_t l, armpl_int_t nb,
                            const armpl_singlecomplex_t *v, armpl_int_t ldv,
                            const armpl_singlecomplex_t *t, armpl_int_t ldt,
                            armpl_singlecomplex_t *a, armpl_int_t lda,
                            armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpmqrt*, *LAPACKE_dtpmqrt*, *LAPACKE_stpmqrt* and *LAPACKE_ztpmqrt*. It also exists with a native Fortran interface as *ctpmqrt*.

4.19.552 LAPACKE_ctpmqrt_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctpmqrt_work(armpl_int_t matrix_layout, char side,
                                 char trans, armpl_int_t m, armpl_int_t n,
                                 armpl_int_t k, armpl_int_t l, armpl_int_t nb,
                                 const armpl_singlecomplex_t *v,
                                 armpl_int_t ldv,
                                 const armpl_singlecomplex_t *t,
                                 armpl_int_t ldt, armpl_singlecomplex_t *a,
                                 armpl_int_t lda, armpl_singlecomplex_t *b,
                                 armpl_int_t ldb,
                                 armpl_singlecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpmqrt_work*, *LAPACKE_dtpmqrt_work*, *LAPACKE_stpmqrt_work* and *LAPACKE_ztpmqrt_work*.

4.19.553 LAPACKE_ctpqrt

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctpqrt(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t l, armpl_int_t nb,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           armpl_singlecomplex_t *b, armpl_int_t ldb,
                           armpl_singlecomplex_t *t, armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.554 LAPACKE_ctpqrt2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctpqrt2(armpl_int_t matrix_layout, armpl_int_t m,
                            armpl_int_t n, armpl_int_t l,
                            armpl_singlecomplex_t *a, armpl_int_t lda,
                            armpl_singlecomplex_t *b, armpl_int_t ldb,
                            armpl_singlecomplex_t *t, armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.555 LAPACKE_ctpqrt2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctpqrt2_work(armpl_int_t matrix_layout, armpl_int_t m,
                                 armpl_int_t n, armpl_int_t l,
                                 armpl_singlecomplex_t *a, armpl_int_t lda,
                                 armpl_singlecomplex_t *b, armpl_int_t ldb,
                                 armpl_singlecomplex_t *t, armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpqrt2_work*, *LAPACKE_dtpqrt2_work*, *LAPACKE_stpqrt2_work* and *LAPACKE_ztpqrt2_work*.

4.19.556 LAPACKE_ctpqrt_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctpqrt_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t l, armpl_int_t nb,
                                armpl_singlecomplex_t *a, armpl_int_t lda,
                                armpl_singlecomplex_t *b, armpl_int_t ldb,
                                armpl_singlecomplex_t *t, armpl_int_t ldt,
                                armpl_singlecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpqrt_work*, *LAPACKE_dtpqrt_work*, *LAPACKE_stpqrt_work* and *LAPACKE_ztpqrt_work*.

4.19.557 LAPACKE_ctprfb

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctprfb(armpl_int_t matrix_layout, char side, char trans,
                           char direct, char storev, armpl_int_t m,
                           armpl_int_t n, armpl_int_t k, armpl_int_t l,
                           const armpl_singlecomplex_t *v, armpl_int_t ldv,
                           const armpl_singlecomplex_t *t, armpl_int_t ldt,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctprfb*, *LAPACKE_dtprfb*, *LAPACKE_stprfb* and *LAPACKE_ztprfb*. It also exists with a native Fortran interface as *ctprfb*.

4.19.558 LAPACKE_ctprfb_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctprfb_work(armpl_int_t matrix_layout, char side,
                                char trans, char direct, char storev,
                                armpl_int_t m, armpl_int_t n, armpl_int_t k,
                                armpl_int_t l, const armpl_singlecomplex_t *v,
                                armpl_int_t ldv,
                                const armpl_singlecomplex_t *t,
                                armpl_int_t ldt, armpl_singlecomplex_t *a,
                                armpl_int_t lda, armpl_singlecomplex_t *b,
                                armpl_int_t ldb, armpl_singlecomplex_t *work,
                                armpl_int_t ldwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctprfb_work*, *LAPACKE_dtprfb_work*, *LAPACKE_stprfb_work* and *LAPACKE_ztprfb_work*.

4.19.559 LAPACKE_ctprfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctprfs(armpl_int_t matrix_layout, char uplo, char trans,
                           char diag, armpl_int_t n, armpl_int_t nrhs,
                           const armpl_singlecomplex_t *ap,
                           const armpl_singlecomplex_t *b, armpl_int_t ldb,
                           const armpl_singlecomplex_t *x, armpl_int_t ldx,
                           float *ferr, float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctprfs*, *LAPACKE_dtprfs*, *LAPACKE_stprfs* and *LAPACKE_ztprfs*. It also exists with a native Fortran interface as *ctprfs*.

4.19.560 LAPACKE_ctprfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctprfs_work(armpl_int_t matrix_layout, char uplo,
                                char trans, char diag, armpl_int_t n,
                                armpl_int_t nrhs,
                                const armpl_singlecomplex_t *ap,
                                const armpl_singlecomplex_t *b,
                                armpl_int_t ldb,
                                const armpl_singlecomplex_t *x,
                                armpl_int_t ldx, float *ferr, float *berr,
                                armpl_singlecomplex_t *work, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctprfs_work*, *LAPACKE_dtprfs_work*, *LAPACKE_stprfs_work* and *LAPACKE_ztprfs_work*.

4.19.561 LAPACKE_ctptri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctptri(armpl_int_t matrix_layout, char uplo, char diag,
                           armpl_int_t n, armpl_singlecomplex_t *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctptri*, *LAPACKE_dtptri*, *LAPACKE_stptri* and *LAPACKE_ztptri*. It also exists with a native Fortran interface as *ctptri*.

4.19.562 LAPACKE_ctptri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctptri_work(armpl_int_t matrix_layout, char uplo,
                                char diag, armpl_int_t n,
                                armpl_singlecomplex_t *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctptri_work*, *LAPACKE_dtptri_work*, *LAPACKE_stptri_work* and *LAPACKE_ztptri_work*.

4.19.563 LAPACKE_ctptrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctptrs(armpl_int_t matrix_layout, char uplo, char trans,
                           char diag, armpl_int_t n, armpl_int_t nrhs,
                           const armpl_singlecomplex_t *ap,
                           armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctptrs*, *LAPACKE_dtptrs*, *LAPACKE_stptrs* and *LAPACKE_ztptrs*. It also exists with a native Fortran interface as *ctptrs*.

4.19.564 LAPACKE_ctptrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctptrs_work(armpl_int_t matrix_layout, char uplo,
                                char trans, char diag, armpl_int_t n,
                                armpl_int_t nrhs,
                                const armpl_singlecomplex_t *ap,
                                armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctptrs_work*, *LAPACKE_dtptrs_work*, *LAPACKE_stptrs_work* and *LAPACKE_ztptrs_work*.

4.19.565 LAPACKE_ctpttf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctpttf(armpl_int_t matrix_layout, char transr, char uplo,
                           armpl_int_t n, const armpl_singlecomplex_t *ap,
                           armpl_singlecomplex_t *arf);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpttf*, *LAPACKE_dtpttf*, *LAPACKE_stpttf* and *LAPACKE_ztpttf*. It also exists with a native Fortran interface as *ctpttf*.

4.19.566 LAPACKE_ctpttf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctpttf_work(armpl_int_t matrix_layout, char transr,
                                char uplo, armpl_int_t n,
                                const armpl_singlecomplex_t *ap,
                                armpl_singlecomplex_t *arf);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpttf_work*, *LAPACKE_dtpttf_work*, *LAPACKE_stpttf_work* and *LAPACKE_ztpttf_work*.

4.19.567 LAPACKE_ctpttr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctpttr(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, const armpl_singlecomplex_t *ap,
                           armpl_singlecomplex_t *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpttr*, *LAPACKE_dtpttr*, *LAPACKE_stpttr* and *LAPACKE_ztpttr*. It also exists with a native Fortran interface as *ctpttr*.

4.19.568 LAPACKE_ctpttr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctpttr_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n,
                                const armpl_singlecomplex_t *ap,
                                armpl_singlecomplex_t *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpttr_work*, *LAPACKE_dtpttr_work*, *LAPACKE_stpttr_work* and *LAPACKE_ztpttr_work*.

4.19.569 LAPACKE_ctrcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctrcon(armpl_int_t matrix_layout, char norm, char uplo,
                           char diag, armpl_int_t n,
                           const armpl_singlecomplex_t *a, armpl_int_t lda,
                           float *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrcon*, *LAPACKE_dtrcon*, *LAPACKE_strcon* and *LAPACKE_ztrcon*. It also exists with a native Fortran interface as *ctrcon*.

4.19.570 LAPACKE_ctrcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctrcon_work(armpl_int_t matrix_layout, char norm,
                                char uplo, char diag, armpl_int_t n,
                                const armpl_singlecomplex_t *a,
                                armpl_int_t lda, float *rcond,
                                armpl_singlecomplex_t *work, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrcon_work*, *LAPACKE_dtrcon_work*, *LAPACKE_strcon_work* and *LAPACKE_ztrcon_work*.

4.19.571 LAPACKE_ctrevc

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctrevc(armpl_int_t matrix_layout, char side, char howmny,
                           const armpl_int_t *select, armpl_int_t n,
                           armpl_singlecomplex_t *t, armpl_int_t ldt,
                           armpl_singlecomplex_t *vl, armpl_int_t ldvl,
                           armpl_singlecomplex_t *vr, armpl_int_t ldvr,
                           armpl_int_t mm, armpl_int_t *m);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrevc*, *LAPACKE_dtrevc*, *LAPACKE_strevc* and *LAPACKE_ztrevc*. It also exists with a native Fortran interface as *ctrevc*.

4.19.572 LAPACKE_ctrevc_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctrevc_work(armpl_int_t matrix_layout, char side,
                                char howmny, const armpl_int_t *select,
                                armpl_int_t n, armpl_singlecomplex_t *t,
                                armpl_int_t ldt, armpl_singlecomplex_t *vl,
                                armpl_int_t ldvl, armpl_singlecomplex_t *vr,
                                armpl_int_t ldvr, armpl_int_t mm,
                                armpl_int_t *m, armpl_singlecomplex_t *work,
                                float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrevc_work*, *LAPACKE_dtrevc_work*, *LAPACKE_strevc_work* and *LAPACKE_ztrevc_work*.

4.19.573 LAPACKE_ctrexc

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctrexc(armpl_int_t matrix_layout, char compq,
                           armpl_int_t n, armpl_singlecomplex_t *t,
                           armpl_int_t ldt, armpl_singlecomplex_t *q,
                           armpl_int_t ldq, armpl_int_t ifst,
                           armpl_int_t ilst);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrexc*, *LAPACKE_dtrexc*, *LAPACKE_strexc* and *LAPACKE_ztrexc*. It also exists with a native Fortran interface as *ctrexc*.

4.19.574 LAPACKE_ctrexc_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctrexc_work(armpl_int_t matrix_layout, char compq,
                                armpl_int_t n, armpl_singlecomplex_t *t,
                                armpl_int_t ldt, armpl_singlecomplex_t *q,
                                armpl_int_t ldq, armpl_int_t ifst,
                                armpl_int_t ilst);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrexc_work*, *LAPACKE_dtrexc_work*, *LAPACKE_strexc_work* and *LAPACKE_ztrexc_work*.

4.19.575 LAPACKE_ctrrfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctrrfs(armpl_int_t matrix_layout, char uplo, char trans,
                           char diag, armpl_int_t n, armpl_int_t nrhs,
                           const armpl_singlecomplex_t *a, armpl_int_t lda,
                           const armpl_singlecomplex_t *b, armpl_int_t ldb,
                           const armpl_singlecomplex_t *x, armpl_int_t ldx,
                           float *ferr, float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrrfs*, *LAPACKE_dtrrfs*, *LAPACKE_strrfs* and *LAPACKE_ztrrfs*. It also exists with a native Fortran interface as *ctrrfs*.

4.19.576 LAPACKE_ctrrfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctrrfs_work(armpl_int_t matrix_layout, char uplo,
                                char trans, char diag, armpl_int_t n,
                                armpl_int_t nrhs,
```

(continues on next page)

(continued from previous page)

```
                                const armpl_singlecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_singlecomplex_t *b,
                                armpl_int_t ldb,
                                const armpl_singlecomplex_t *x,
                                armpl_int_t ldx, float *ferr, float *berr,
                                armpl_singlecomplex_t *work, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrrfs_work*, *LAPACKE_dtrrfs_work*, *LAPACKE_strrfs_work* and *LAPACKE_ztrrfs_work*.

4.19.577 LAPACKE_ctrsen

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctrsen(armpl_int_t matrix_layout, char job, char compq,
                           const armpl_int_t *select, armpl_int_t n,
                           armpl_singlecomplex_t *t, armpl_int_t ldt,
                           armpl_singlecomplex_t *q, armpl_int_t ldq,
                           armpl_singlecomplex_t *w, armpl_int_t *m, float *s,
                           float *sep);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrsen*, *LAPACKE_dtrsen*, *LAPACKE_strsen* and *LAPACKE_ztrsen*. It also exists with a native Fortran interface as *ctrsen*.

4.19.578 LAPACKE_ctrsen_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctrsen_work(armpl_int_t matrix_layout, char job,
                                char compq, const armpl_int_t *select,
                                armpl_int_t n, armpl_singlecomplex_t *t,
                                armpl_int_t ldt, armpl_singlecomplex_t *q,
                                armpl_int_t ldq, armpl_singlecomplex_t *w,
                                armpl_int_t *m, float *s, float *sep,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrsen_work*, *LAPACKE_dtrsen_work*, *LAPACKE_strsen_work* and *LAPACKE_ztrsen_work*.

4.19.579 LAPACKE_ctrsna

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctrsna(armpl_int_t matrix_layout, char job, char howmny,
                           const armpl_int_t *select, armpl_int_t n,
                           const armpl_singlecomplex_t *t, armpl_int_t ldt,
                           const armpl_singlecomplex_t *vl, armpl_int_t ldvl,
                           const armpl_singlecomplex_t *vr, armpl_int_t ldvr,
                           float *s, float *sep, armpl_int_t mm,
                           armpl_int_t *m);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrsna*, *LAPACKE_dtrsna*, *LAPACKE_strsna* and *LAPACKE_ztrsna*. It also exists with a native Fortran interface as *ctrsna*.

4.19.580 LAPACKE_ctrsna_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctrsna_work(armpl_int_t matrix_layout, char job,
                                char howmny, const armpl_int_t *select,
                                armpl_int_t n, const armpl_singlecomplex_t *t,
                                armpl_int_t ldt,
                                const armpl_singlecomplex_t *vl,
                                armpl_int_t ldvl,
                                const armpl_singlecomplex_t *vr,
                                armpl_int_t ldvr, float *s, float *sep,
                                armpl_int_t mm, armpl_int_t *m,
                                armpl_singlecomplex_t *work,
                                armpl_int_t ldwork, float *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrsna_work*, *LAPACKE_dtrsna_work*, *LAPACKE_strsna_work* and *LAPACKE_ztrsna_work*.

4.19.581 LAPACKE_ctrsyl

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctrsyl(armpl_int_t matrix_layout, char trana, char tranb,
                           armpl_int_t isgn, armpl_int_t m, armpl_int_t n,
                           const armpl_singlecomplex_t *a, armpl_int_t lda,
                           const armpl_singlecomplex_t *b, armpl_int_t ldb,
                           armpl_singlecomplex_t *c, armpl_int_t ldc,
                           float *scale);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrsyl*, *LAPACKE_dtrsyl*, *LAPACKE_strsyl* and *LAPACKE_ztrsyl*. It also exists with a native Fortran interface as *ctrsyl*.

4.19.582 LAPACKE_ctrsyl_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctrsyl_work(armpl_int_t matrix_layout, char trana,
                                char tranb, armpl_int_t isgn, armpl_int_t m,
                                armpl_int_t n, const armpl_singlecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_singlecomplex_t *b,
                                armpl_int_t ldb, armpl_singlecomplex_t *c,
                                armpl_int_t ldc, float *scale);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrsyl_work*, *LAPACKE_dtrsyl_work*, *LAPACKE_strsyl_work* and *LAPACKE_ztrsyl_work*.

4.19.583 LAPACKE_ctrtri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctrtri(armpl_int_t matrix_layout, char uplo, char diag,
                           armpl_int_t n, armpl_singlecomplex_t *a,
                           armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrtri*, *LAPACKE_dtrtri*, *LAPACKE_strtri* and *LAPACKE_ztrtri*. It also exists with a native Fortran interface as *ctrtri*.

4.19.584 LAPACKE_ctrtri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctrtri_work(armpl_int_t matrix_layout, char uplo,
                                char diag, armpl_int_t n,
                                armpl_singlecomplex_t *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrtri_work*, *LAPACKE_dtrtri_work*, *LAPACKE_strtri_work* and *LAPACKE_ztrtri_work*.

4.19.585 LAPACKE_ctrtrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctrtrs(armpl_int_t matrix_layout, char uplo, char trans,
                           char diag, armpl_int_t n, armpl_int_t nrhs,
                           const armpl_singlecomplex_t *a, armpl_int_t lda,
                           armpl_singlecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrtrs*, *LAPACKE_dtrtrs*, *LAPACKE_strtrs* and *LAPACKE_ztrtrs*. It also exists with a native Fortran interface as *ctrtrs*.

4.19.586 LAPACKE_ctrtrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctrtrs_work(armpl_int_t matrix_layout, char uplo,
                                char trans, char diag, armpl_int_t n,
                                armpl_int_t nrhs,
                                const armpl_singlecomplex_t *a,
                                armpl_int_t lda, armpl_singlecomplex_t *b,
                                armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrtrs_work*, *LAPACKE_dtrtrs_work*, *LAPACKE_strtrs_work* and *LAPACKE_ztrtrs_work*.

4.19.587 LAPACKE_ctrttf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctrttf(armpl_int_t matrix_layout, char transr, char uplo,
                           armpl_int_t n, const armpl_singlecomplex_t *a,
                           armpl_int_t lda, armpl_singlecomplex_t *arf);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrttf*, *LAPACKE_dtrttf*, *LAPACKE_strttf* and *LAPACKE_ztrttf*. It also exists with a native Fortran interface as *ctrttf*.

4.19.588 LAPACKE_ctrttf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctrttf_work(armpl_int_t matrix_layout, char transr,
                                char uplo, armpl_int_t n,
                                const armpl_singlecomplex_t *a,
                                armpl_int_t lda, armpl_singlecomplex_t *arf);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrttf_work*, *LAPACKE_dtrttf_work*, *LAPACKE_strttf_work* and *LAPACKE_ztrttf_work*.

4.19.589 LAPACKE_ctrttp

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctrttp(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, const armpl_singlecomplex_t *a,
                           armpl_int_t lda, armpl_singlecomplex_t *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrttp*, *LAPACKE_dtrttp*, *LAPACKE_strttp* and *LAPACKE_ztrttp*. It also exists with a native Fortran interface as *ctrttp*.

4.19.590 LAPACKE_ctrttp_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctrttp_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, const armpl_singlecomplex_t *a,
                                armpl_int_t lda, armpl_singlecomplex_t *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrttp_work*, *LAPACKE_dtrttp_work*, *LAPACKE_strttp_work* and *LAPACKE_ztrttp_work*.

4.19.591 LAPACKE_ctzrzf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctzrzf(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_singlecomplex_t *a,
                           armpl_int_t lda, armpl_singlecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctzrzf*, *LAPACKE_dtzrzf*, *LAPACKE_stzrzf* and *LAPACKE_ztzrzf*. It also exists with a native Fortran interface as *ctzrzf*.

4.19.592 LAPACKE_ctzrzf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ctzrzf_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_singlecomplex_t *a,
                                armpl_int_t lda, armpl_singlecomplex_t *tau,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctzrzf_work*, *LAPACKE_dtzrzf_work*, *LAPACKE_stzrzf_work* and *LAPACKE_ztzrzf_work*.

4.19.593 LAPACKE_cunbdb

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cunbdb(armpl_int_t matrix_layout, char trans, char signs,
                           armpl_int_t m, armpl_int_t p, armpl_int_t q,
                           armpl_singlecomplex_t *x11, armpl_int_t ldx11,
                           armpl_singlecomplex_t *x12, armpl_int_t ldx12,
```

(continues on next page)

(continued from previous page)

```
                                armpl_singlecomplex_t *x21, armpl_int_t ldx21,
                                armpl_singlecomplex_t *x22, armpl_int_t ldx22,
                                float *theta, float *phi,
                                armpl_singlecomplex_t *taup1,
                                armpl_singlecomplex_t *taup2,
                                armpl_singlecomplex_t *tauq1,
                                armpl_singlecomplex_t *tauq2);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunbdb* and *LAPACKE_zunbdb*. It also exists with a native Fortran interface as *cunbdb*.

4.19.594 LAPACKE_cunbdb_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cunbdb_work(armpl_int_t matrix_layout, char trans,
                                char signs, armpl_int_t m, armpl_int_t p,
                                armpl_int_t q, armpl_singlecomplex_t *x11,
                                armpl_int_t ldx11, armpl_singlecomplex_t *x12,
                                armpl_int_t ldx12, armpl_singlecomplex_t *x21,
                                armpl_int_t ldx21, armpl_singlecomplex_t *x22,
                                armpl_int_t ldx22, float *theta, float *phi,
                                armpl_singlecomplex_t *taup1,
                                armpl_singlecomplex_t *taup2,
                                armpl_singlecomplex_t *tauq1,
                                armpl_singlecomplex_t *tauq2,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunbdb_work* and *LAPACKE_zunbdb_work*.

4.19.595 LAPACKE_cuncsd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cuncsd(armpl_int_t matrix_layout, char jobu1, char jobu2,
                           char jobv1t, char jobv2t, char trans, char signs,
                           armpl_int_t m, armpl_int_t p, armpl_int_t q,
                           armpl_singlecomplex_t *x11, armpl_int_t ldx11,
                           armpl_singlecomplex_t *x12, armpl_int_t ldx12,
                           armpl_singlecomplex_t *x21, armpl_int_t ldx21,
                           armpl_singlecomplex_t *x22, armpl_int_t ldx22,
                           float *theta, armpl_singlecomplex_t *u1,
                           armpl_int_t ldu1, armpl_singlecomplex_t *u2,
                           armpl_int_t ldu2, armpl_singlecomplex_t *v1t,
                           armpl_int_t ldv1t, armpl_singlecomplex_t *v2t,
                           armpl_int_t ldv2t);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cuncsd* and *LAPACKE_zuncsd*. It also exists with a native Fortran interface as *cuncsd*.

4.19.596 LAPACKE_cuncsd2by1

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cuncsd2by1(armpl_int_t matrix_layout, char jobu1,
                               char jobu2, char jobv1t, armpl_int_t m,
                               armpl_int_t p, armpl_int_t q,
                               armpl_singlecomplex_t *x11, armpl_int_t ldx11,
                               armpl_singlecomplex_t *x21, armpl_int_t ldx21,
                               float *theta, armpl_singlecomplex_t *u1,
                               armpl_int_t ldu1, armpl_singlecomplex_t *u2,
                               armpl_int_t ldu2, armpl_singlecomplex_t *v1t,
                               armpl_int_t ldv1t);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cuncsd2by1* and *LAPACKE_zuncsd2by1*. It also exists with a native Fortran interface as *cuncsd2by1*.

4.19.597 LAPACKE_cuncsd2by1_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cuncsd2by1_work(armpl_int_t matrix_layout, char jobu1,
                                    char jobu2, char jobv1t, armpl_int_t m,
                                    armpl_int_t p, armpl_int_t q,
                                    armpl_singlecomplex_t *x11,
                                    armpl_int_t ldx11,
                                    armpl_singlecomplex_t *x21,
                                    armpl_int_t ldx21, float *theta,
                                    armpl_singlecomplex_t *u1,
                                    armpl_int_t ldu1,
                                    armpl_singlecomplex_t *u2,
                                    armpl_int_t ldu2,
                                    armpl_singlecomplex_t *v1t,
                                    armpl_int_t ldv1t,
                                    armpl_singlecomplex_t *work,
                                    armpl_int_t lwork, float *rwork,
                                    armpl_int_t lrwork, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cuncsd2by1_work* and *LAPACKE_zuncsd2by1_work*.

4.19.598 LAPACKE_cuncsd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cuncsd_work(armpl_int_t matrix_layout, char jobu1,
                                char jobu2, char jobv1t, char jobv2t,
                                char trans, char signs, armpl_int_t m,
                                armpl_int_t p, armpl_int_t q,
                                armpl_singlecomplex_t *x11, armpl_int_t ldx11,
                                armpl_singlecomplex_t *x12, armpl_int_t ldx12,
                                armpl_singlecomplex_t *x21, armpl_int_t ldx21,
                                armpl_singlecomplex_t *x22, armpl_int_t ldx22,
                                float *theta, armpl_singlecomplex_t *u1,
                                armpl_int_t ldu1, armpl_singlecomplex_t *u2,
                                armpl_int_t ldu2, armpl_singlecomplex_t *v1t,
                                armpl_int_t ldv1t, armpl_singlecomplex_t *v2t,
                                armpl_int_t ldv2t,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork, float *rwork,
                                armpl_int_t lrwork, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cuncsd_work* and *LAPACKE_zuncsd_work*.

4.19.599 LAPACKE_cungbr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cungbr(armpl_int_t matrix_layout, char vect,
                           armpl_int_t m, armpl_int_t n, armpl_int_t k,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           const armpl_singlecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cungbr* and *LAPACKE_zungbr*. It also exists with a native Fortran interface as *cungbr*.

4.19.600 LAPACKE_cungbr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cungbr_work(armpl_int_t matrix_layout, char vect,
                                armpl_int_t m, armpl_int_t n, armpl_int_t k,
                                armpl_singlecomplex_t *a, armpl_int_t lda,
                                const armpl_singlecomplex_t *tau,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cungbr_work* and *LAPACKE_zungbr_work*.

4.19.601 LAPACKE_cunghr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cunghr(armpl_int_t matrix_layout, armpl_int_t n,
                           armpl_int_t ilo, armpl_int_t ihi,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           const armpl_singlecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunghr* and *LAPACKE_zunghr*. It also exists with a native Fortran interface as *cunghr*.

4.19.602 LAPACKE_cunghr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cunghr_work(armpl_int_t matrix_layout, armpl_int_t n,
                                armpl_int_t ilo, armpl_int_t ihi,
                                armpl_singlecomplex_t *a, armpl_int_t lda,
                                const armpl_singlecomplex_t *tau,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunghr_work* and *LAPACKE_zunghr_work*.

4.19.603 LAPACKE_cunglq

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cunglq(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t k,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           const armpl_singlecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunglq* and *LAPACKE_zunglq*. It also exists with a native Fortran interface as *cunglq*.

4.19.604 LAPACKE_cunglq_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cunglq_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t k,
                                armpl_singlecomplex_t *a, armpl_int_t lda,
                                const armpl_singlecomplex_t *tau,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunglq_work* and *LAPACKE_zunglq_work*.

4.19.605 LAPACKE_cungql

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cungql(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t k,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           const armpl_singlecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cungql* and *LAPACKE_zungql*. It also exists with a native Fortran interface as *cungql*.

4.19.606 LAPACKE_cungql_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cungql_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t k,
                                armpl_singlecomplex_t *a, armpl_int_t lda,
                                const armpl_singlecomplex_t *tau,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cungql_work* and *LAPACKE_zungql_work*.

4.19.607 LAPACKE_cungqr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cungqr(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t k,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           const armpl_singlecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cungqr* and *LAPACKE_zungqr*. It also exists with a native Fortran interface as *cungqr*.

4.19.608 LAPACKE_cungqr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cungqr_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t k,
                                armpl_singlecomplex_t *a, armpl_int_t lda,
                                const armpl_singlecomplex_t *tau,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cungqr_work* and *LAPACKE_zungqr_work*.

4.19.609 LAPACKE_cungrq

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cungrq(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t k,
                           armpl_singlecomplex_t *a, armpl_int_t lda,
                           const armpl_singlecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cungrq* and *LAPACKE_zungrq*. It also exists with a native Fortran interface as *cungrq*.

4.19.610 LAPACKE_cungrq_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cungrq_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t k,
                                armpl_singlecomplex_t *a, armpl_int_t lda,
                                const armpl_singlecomplex_t *tau,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cungrq_work* and *LAPACKE_zungrq_work*.

4.19.611 LAPACKE_cungtr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cungtr(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_singlecomplex_t *a,
                           armpl_int_t lda,
                           const armpl_singlecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cungtr* and *LAPACKE_zungtr*. It also exists with a native Fortran interface as *cungtr*.

4.19.612 LAPACKE_cungtr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cungtr_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_singlecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_singlecomplex_t *tau,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cungtr_work* and *LAPACKE_zungtr_work*.

4.19.613 LAPACKE_cunmbr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cunmbr(armpl_int_t matrix_layout, char vect, char side,
                           char trans, armpl_int_t m, armpl_int_t n,
                           armpl_int_t k, const armpl_singlecomplex_t *a,
                           armpl_int_t lda, const armpl_singlecomplex_t *tau,
                           armpl_singlecomplex_t *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunmbr* and *LAPACKE_zunmbr*. It also exists with a native Fortran interface as *cunmbr*.

4.19.614 LAPACKE_cunmbr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cunmbr_work(armpl_int_t matrix_layout, char vect,
                                char side, char trans, armpl_int_t m,
                                armpl_int_t n, armpl_int_t k,
                                const armpl_singlecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_singlecomplex_t *tau,
                                armpl_singlecomplex_t *c, armpl_int_t ldc,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunmbr_work* and *LAPACKE_zunmbr_work*.

4.19.615 LAPACKE_cunmhr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cunmhr(armpl_int_t matrix_layout, char side, char trans,
                           armpl_int_t m, armpl_int_t n, armpl_int_t ilo,
                           armpl_int_t ihi, const armpl_singlecomplex_t *a,
                           armpl_int_t lda, const armpl_singlecomplex_t *tau,
                           armpl_singlecomplex_t *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunmhr* and *LAPACKE_zunmhr*. It also exists with a native Fortran interface as *cunmhr*.

4.19.616 LAPACKE_cunmhr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cunmhr_work(armpl_int_t matrix_layout, char side,
                                char trans, armpl_int_t m, armpl_int_t n,
                                armpl_int_t ilo, armpl_int_t ihi,
                                const armpl_singlecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_singlecomplex_t *tau,
                                armpl_singlecomplex_t *c, armpl_int_t ldc,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunmhr_work* and *LAPACKE_zunmhr_work*.

4.19.617 LAPACKE_cunmlq

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cunmlq(armpl_int_t matrix_layout, char side, char trans,
                           armpl_int_t m, armpl_int_t n, armpl_int_t k,
                           const armpl_singlecomplex_t *a, armpl_int_t lda,
                           const armpl_singlecomplex_t *tau,
                           armpl_singlecomplex_t *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunmlq* and *LAPACKE_zunmlq*. It also exists with a native Fortran interface as *cunmlq*.

4.19.618 LAPACKE_cunmlq_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cunmlq_work(armpl_int_t matrix_layout, char side,
                                char trans, armpl_int_t m, armpl_int_t n,
                                armpl_int_t k, const armpl_singlecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_singlecomplex_t *tau,
                                armpl_singlecomplex_t *c, armpl_int_t ldc,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunmlq_work* and *LAPACKE_zunmlq_work*.

4.19.619 LAPACKE_cunmql

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cunmql(armpl_int_t matrix_layout, char side, char trans,
                           armpl_int_t m, armpl_int_t n, armpl_int_t k,
                           const armpl_singlecomplex_t *a, armpl_int_t lda,
                           const armpl_singlecomplex_t *tau,
                           armpl_singlecomplex_t *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunmql* and *LAPACKE_zunmql*. It also exists with a native Fortran interface as *cunmql*.

4.19.620 LAPACKE_cunmql_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cunmql_work(armpl_int_t matrix_layout, char side,
                                char trans, armpl_int_t m, armpl_int_t n,
                                armpl_int_t k, const armpl_singlecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_singlecomplex_t *tau,
                                armpl_singlecomplex_t *c, armpl_int_t ldc,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunmql_work* and *LAPACKE_zunmql_work*.

4.19.621 LAPACKE_cunmqr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cunmqr(armpl_int_t matrix_layout, char side, char trans,
                           armpl_int_t m, armpl_int_t n, armpl_int_t k,
                           const armpl_singlecomplex_t *a, armpl_int_t lda,
                           const armpl_singlecomplex_t *tau,
                           armpl_singlecomplex_t *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunmqr* and *LAPACKE_zunmqr*. It also exists with a native Fortran interface as *cunmqr*.

4.19.622 LAPACKE_cunmqr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cunmqr_work(armpl_int_t matrix_layout, char side,
                                char trans, armpl_int_t m, armpl_int_t n,
                                armpl_int_t k, const armpl_singlecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_singlecomplex_t *tau,
                                armpl_singlecomplex_t *c, armpl_int_t ldc,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunmqr_work* and *LAPACKE_zunmqr_work*.

4.19.623 LAPACKE_cunmrq

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cunmrq(armpl_int_t matrix_layout, char side, char trans,
                           armpl_int_t m, armpl_int_t n, armpl_int_t k,
                           const armpl_singlecomplex_t *a, armpl_int_t lda,
                           const armpl_singlecomplex_t *tau,
                           armpl_singlecomplex_t *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunmrq* and *LAPACKE_zunmrq*. It also exists with a native Fortran interface as *cunmrq*.

4.19.624 LAPACKE_cunmrq_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cunmrq_work(armpl_int_t matrix_layout, char side,
                                char trans, armpl_int_t m, armpl_int_t n,
                                armpl_int_t k, const armpl_singlecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_singlecomplex_t *tau,
```

(continues on next page)

(continued from previous page)

```
                                armpl_singlecomplex_t *c, armpl_int_t ldc,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunmrq_work* and *LAPACKE_zunmrq_work*.

4.19.625 LAPACKE_cunmrz

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cunmrz(armpl_int_t matrix_layout, char side, char trans,
                           armpl_int_t m, armpl_int_t n, armpl_int_t k,
                           armpl_int_t l, const armpl_singlecomplex_t *a,
                           armpl_int_t lda, const armpl_singlecomplex_t *tau,
                           armpl_singlecomplex_t *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunmrz* and *LAPACKE_zunmrz*. It also exists with a native Fortran interface as *cunmrz*.

4.19.626 LAPACKE_cunmrz_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cunmrz_work(armpl_int_t matrix_layout, char side,
                                char trans, armpl_int_t m, armpl_int_t n,
                                armpl_int_t k, armpl_int_t l,
                                const armpl_singlecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_singlecomplex_t *tau,
                                armpl_singlecomplex_t *c, armpl_int_t ldc,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunmrz_work* and *LAPACKE_zunmrz_work*.

4.19.627 LAPACKE_cunmtr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cunmtr(armpl_int_t matrix_layout, char side, char uplo,
                           char trans, armpl_int_t m, armpl_int_t n,
                           const armpl_singlecomplex_t *a, armpl_int_t lda,
                           const armpl_singlecomplex_t *tau,
                           armpl_singlecomplex_t *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunmtr* and *LAPACKE_zunmtr*. It also exists with a native Fortran interface as *cunmtr*.

4.19.628 LAPACKE_cunmtr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cunmtr_work(armpl_int_t matrix_layout, char side,
                                char uplo, char trans, armpl_int_t m,
                                armpl_int_t n, const armpl_singlecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_singlecomplex_t *tau,
                                armpl_singlecomplex_t *c, armpl_int_t ldc,
                                armpl_singlecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.629 LAPACKE_cupgtr

4.19.630 LAPACKE_cupgtr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cupgtr_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n,
                                const armpl_singlecomplex_t *ap,
                                const armpl_singlecomplex_t *tau,
                                armpl_singlecomplex_t *q, armpl_int_t ldq,
                                armpl_singlecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cupgtr_work* and *LAPACKE_zupgtr_work*.

4.19.631 LAPACKE_cupmtr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cupmtr(armpl_int_t matrix_layout, char side, char uplo,
                           char trans, armpl_int_t m, armpl_int_t n,
                           const armpl_singlecomplex_t *ap,
                           const armpl_singlecomplex_t *tau,
                           armpl_singlecomplex_t *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cupmtr* and *LAPACKE_zupmtr*. It also exists with a native Fortran interface as *cupmtr*.

4.19.632 LAPACKE_cupmtr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_cupmtr_work(armpl_int_t matrix_layout, char side,
                                char uplo, char trans, armpl_int_t m,
                                armpl_int_t n,
                                const armpl_singlecomplex_t *ap,
                                const armpl_singlecomplex_t *tau,
                                armpl_singlecomplex_t *c, armpl_int_t ldc,
                                armpl_singlecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cupmtr_work* and *LAPACKE_zupmtr_work*.

4.19.633 LAPACKE_dbbcsd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dbbcsd(armpl_int_t matrix_layout, char jobu1, char jobu2,
                           char jobv1t, char jobv2t, char trans,
                           armpl_int_t m, armpl_int_t p, armpl_int_t q,
                           double *theta, double *phi, double *u1,
                           armpl_int_t ldu1, double *u2, armpl_int_t ldu2,
                           double *v1t, armpl_int_t ldv1t, double *v2t,
                           armpl_int_t ldv2t, double *b11d, double *b11e,
                           double *b12d, double *b12e, double *b21d,
                           double *b21e, double *b22d, double *b22e);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cbbcsd*, *LAPACKE_dbbcsd*, *LAPACKE_sbbcsd* and *LAPACKE_zbbcsd*. It also exists with a native Fortran interface as *dbbcsd*.

4.19.634 LAPACKE_dbbcsd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dbbcsd_work(armpl_int_t matrix_layout, char jobu1,
                                char jobu2, char jobv1t, char jobv2t,
                                char trans, armpl_int_t m, armpl_int_t p,
                                armpl_int_t q, double *theta, double *phi,
                                double *u1, armpl_int_t ldu1, double *u2,
                                armpl_int_t ldu2, double *v1t,
                                armpl_int_t ldv1t, double *v2t,
                                armpl_int_t ldv2t, double *b11d, double *b11e,
```

(continues on next page)

(continued from previous page)

```
                                    double *b12d, double *b12e, double *b21d,
                                    double *b21e, double *b22d, double *b22e,
                                    double *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cbbcsd_work*, *LAPACKE_dbbcsd_work*, *LAPACKE_sbbcsd_work* and *LAPACKE_zbbcsd_work*.

4.19.635 LAPACKE_dbdsdc

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dbdsdc(armpl_int_t matrix_layout, char uplo, char compq,
                           armpl_int_t n, double *d, double *e, double *u,
                           armpl_int_t ldu, double *vt, armpl_int_t ldvt,
                           double *q, armpl_int_t *iq);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dbdsdc* and *LAPACKE_sbdsdc*. It also exists with a native Fortran interface as *dbdsdc*.

4.19.636 LAPACKE_dbdsdc_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dbdsdc_work(armpl_int_t matrix_layout, char uplo,
                                char compq, armpl_int_t n, double *d,
                                double *e, double *u, armpl_int_t ldu,
                                double *vt, armpl_int_t ldvt, double *q,
                                armpl_int_t *iq, double *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dbdsdc_work* and *LAPACKE_sbdsdc_work*.

4.19.637 LAPACKE_dbdsqr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dbdsqr(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t ncvt, armpl_int_t nru,
                           armpl_int_t ncc, double *d, double *e, double *vt,
                           armpl_int_t ldvt, double *u, armpl_int_t ldu,
                           double *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cbdsqr*, *LAPACKE_dbdsqr*, *LAPACKE_sbdsqr* and *LAPACKE_zbdsqr*. It also exists with a native Fortran interface as *dbdsqr*.

4.19.638 LAPACKE_dbdsqr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dbdsqr_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t ncvt,
                                armpl_int_t nru, armpl_int_t ncc, double *d,
                                double *e, double *vt, armpl_int_t ldvt,
                                double *u, armpl_int_t ldu, double *c,
                                armpl_int_t ldc, double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cbdsqr_work*, *LAPACKE_dbdsqr_work*, *LAPACKE_sbdsqr_work* and *LAPACKE_zbdsqr_work*.

4.19.639 LAPACKE_dbdsvdx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dbdsvdx(armpl_int_t matrix_layout, char uplo, char jobz,
                            char range, armpl_int_t n, double *d, double *e,
                            double vl, double vu, armpl_int_t il,
                            armpl_int_t iu, armpl_int_t *ns, double *s,
                            double *z, armpl_int_t ldz, armpl_int_t *superb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dbdsvdx* and *LAPACKE_sbdsvdx*. It also exists with a native Fortran interface as *dbdsvdx*.

4.19.640 LAPACKE_dbdsvdx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dbdsvdx_work(armpl_int_t matrix_layout, char uplo,
                                 char jobz, char range, armpl_int_t n,
                                 double *d, double *e, double vl, double vu,
                                 armpl_int_t il, armpl_int_t iu,
                                 armpl_int_t *ns, double *s, double *z,
                                 armpl_int_t ldz, double *work,
                                 armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dbdsvdx_work* and *LAPACKE_sbdsvdx_work*.

4.19.641 LAPACKE_ddisna

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ddisna(char job, armpl_int_t m, armpl_int_t n,
                           const double *d, double *sep);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ddisna* and *LAPACKE_sdisna*. It also exists with a native Fortran interface as *ddisna*.

4.19.642 LAPACKE_ddisna_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ddisna_work(char job, armpl_int_t m, armpl_int_t n,
                                const double *d, double *sep);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ddisna_work* and *LAPACKE_sdisna_work*.

4.19.643 LAPACKE_dgbbrd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgbbrd(armpl_int_t matrix_layout, char vect,
                           armpl_int_t m, armpl_int_t n, armpl_int_t ncc,
                           armpl_int_t kl, armpl_int_t ku, double *ab,
                           armpl_int_t ldab, double *d, double *e, double *q,
                           armpl_int_t ldq, double *pt, armpl_int_t ldpt,
                           double *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbbrd*, *LAPACKE_dgbbrd*, *LAPACKE_sgbbrd* and *LAPACKE_zgbbrd*. It also exists with a native Fortran interface as *dgbbrd*.

4.19.644 LAPACKE_dgbbrd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgbbrd_work(armpl_int_t matrix_layout, char vect,
                                armpl_int_t m, armpl_int_t n, armpl_int_t ncc,
                                armpl_int_t kl, armpl_int_t ku, double *ab,
                                armpl_int_t ldab, double *d, double *e,
                                double *q, armpl_int_t ldq, double *pt,
                                armpl_int_t ldpt, double *c, armpl_int_t ldc,
                                double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbbrd_work*, *LAPACKE_dgbbrd_work*, *LAPACKE_sgbbrd_work* and *LAPACKE_zgbbrd_work*.

4.19.645 LAPACKE_dgbcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgbcon(armpl_int_t matrix_layout, char norm,
                           armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                           const double *ab, armpl_int_t ldab,
                           const armpl_int_t *ipiv, double anorm,
                           double *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbcon*, *LAPACKE_dgbcon*, *LAPACKE_sgbcon* and *LAPACKE_zgbcon*. It also exists with a native Fortran interface as *dgbcon*.

4.19.646 LAPACKE_dgbcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgbcon_work(armpl_int_t matrix_layout, char norm,
                                armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                                const double *ab, armpl_int_t ldab,
                                const armpl_int_t *ipiv, double anorm,
                                double *rcond, double *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbcon_work*, *LAPACKE_dgbcon_work*, *LAPACKE_sgbcon_work* and *LAPACKE_zgbcon_work*.

4.19.647 LAPACKE_dgbequ

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgbequ(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                           const double *ab, armpl_int_t ldab, double *r,
                           double *c, double *rowcnd, double *colcnd,
                           double *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbequ*, *LAPACKE_dgbequ*, *LAPACKE_sgbequ* and *LAPACKE_zgbequ*. It also exists with a native Fortran interface as *dgbequ*.

4.19.648 LAPACKE_dgbequ_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgbequ_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                                const double *ab, armpl_int_t ldab, double *r,
                                double *c, double *rowcnd, double *colcnd,
                                double *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbequ_work*, *LAPACKE_dgbequ_work*, *LAPACKE_sgbequ_work* and *LAPACKE_zgbequ_work*.

4.19.649 LAPACKE_dgbequb

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgbequb(armpl_int_t matrix_layout, armpl_int_t m,
                            armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                            const double *ab, armpl_int_t ldab, double *r,
                            double *c, double *rowcnd, double *colcnd,
                            double *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbequb*, *LAPACKE_dgbequb*, *LAPACKE_sgbequb* and *LAPACKE_zgbequb*. It also exists with a native Fortran interface as *dgbequb*.

4.19.650 LAPACKE_dgbequb_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgbequb_work(armpl_int_t matrix_layout, armpl_int_t m,
                                 armpl_int_t n, armpl_int_t kl,
                                 armpl_int_t ku, const double *ab,
                                 armpl_int_t ldab, double *r, double *c,
                                 double *rowcnd, double *colcnd,
                                 double *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbequb_work*, *LAPACKE_dgbequb_work*, *LAPACKE_sgbequb_work* and *LAPACKE_zgbequb_work*.

4.19.651 LAPACKE_dgbrfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgbrfs(armpl_int_t matrix_layout, char trans,
                           armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                           armpl_int_t nrhs, const double *ab,
```

(continues on next page)

(continued from previous page)

```
                                armpl_int_t ldab, const double *afb,
                                armpl_int_t ldafb, const armpl_int_t *ipiv,
                                const double *b, armpl_int_t ldb, double *x,
                                armpl_int_t ldx, double *ferr, double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbrfs*, *LAPACKE_dgbrfs*, *LAPACKE_sgbrfs* and *LAPACKE_zgbrfs*. It also exists with a native Fortran interface as *dgbrfs*.

4.19.652 LAPACKE_dgbrfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgbrfs_work(armpl_int_t matrix_layout, char trans,
                                armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                                armpl_int_t nrhs, const double *ab,
                                armpl_int_t ldab, const double *afb,
                                armpl_int_t ldafb, const armpl_int_t *ipiv,
                                const double *b, armpl_int_t ldb, double *x,
                                armpl_int_t ldx, double *ferr, double *berr,
                                double *work, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbrfs_work*, *LAPACKE_dgbrfs_work*, *LAPACKE_sgbrfs_work* and *LAPACKE_zgbrfs_work*.

4.19.653 LAPACKE_dgbrfsx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgbrfsx(armpl_int_t matrix_layout, char trans, char equed,
                            armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                            armpl_int_t nrhs, const double *ab,
                            armpl_int_t ldab, const double *afb,
                            armpl_int_t ldafb, const armpl_int_t *ipiv,
                            const double *r, const double *c, const double *b,
                            armpl_int_t ldb, double *x, armpl_int_t ldx,
                            double *rcond, double *berr,
                            armpl_int_t n_err_bnds, double *err_bnds_norm,
                            double *err_bnds_comp, armpl_int_t nparams,
                            double *params);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbrfsx*, *LAPACKE_dgbrfsx*, *LAPACKE_sgbrfsx* and *LAPACKE_zgbrfsx*. It also exists with a native Fortran interface as *dgbrfsx*.

4.19.654 LAPACKE_dgbrfsx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgbrfsx_work(armpl_int_t matrix_layout, char trans,
                                 char equed, armpl_int_t n, armpl_int_t kl,
                                 armpl_int_t ku, armpl_int_t nrhs,
                                 const double *ab, armpl_int_t ldab,
                                 const double *afb, armpl_int_t ldafb,
                                 const armpl_int_t *ipiv, const double *r,
                                 const double *c, const double *b,
                                 armpl_int_t ldb, double *x, armpl_int_t ldx,
                                 double *rcond, double *berr,
                                 armpl_int_t n_err_bnds,
                                 double *err_bnds_norm, double *err_bnds_comp,
                                 armpl_int_t nparams, double *params,
                                 double *work, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbrfsx_work*, *LAPACKE_dgbrfsx_work*, *LAPACKE_sgbrfsx_work* and *LAPACKE_zgbrfsx_work*.

4.19.655 LAPACKE_dgbsv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgbsv(armpl_int_t matrix_layout, armpl_int_t n,
                          armpl_int_t kl, armpl_int_t ku, armpl_int_t nrhs,
                          double *ab, armpl_int_t ldab, armpl_int_t *ipiv,
                          double *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbsv*, *LAPACKE_dgbsv*, *LAPACKE_sgbsv* and *LAPACKE_zgbsv*. It also exists with a native Fortran interface as *dgbsv*.

4.19.656 LAPACKE_dgbsv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgbsv_work(armpl_int_t matrix_layout, armpl_int_t n,
                               armpl_int_t kl, armpl_int_t ku,
                               armpl_int_t nrhs, double *ab, armpl_int_t ldab,
                               armpl_int_t *ipiv, double *b,
                               armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbsv_work*, *LAPACKE_dgbsv_work*, *LAPACKE_sgbsv_work* and *LAPACKE_zgbsv_work*.

4.19.657 LAPACKE_dgbsvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgbsvx(armpl_int_t matrix_layout, char fact, char trans,
                           armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                           armpl_int_t nrhs, double *ab, armpl_int_t ldab,
                           double *afb, armpl_int_t ldafb, armpl_int_t *ipiv,
                           char *equed, double *r, double *c, double *b,
                           armpl_int_t ldb, double *x, armpl_int_t ldx,
                           double *rcond, double *ferr, double *berr,
                           double *rpivot);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbsvx*, *LAPACKE_dgbsvx*, *LAPACKE_sgbsvx* and *LAPACKE_zgbsvx*. It also exists with a native Fortran interface as *dgbsvx*.

4.19.658 LAPACKE_dgbsvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgbsvx_work(armpl_int_t matrix_layout, char fact,
                                char trans, armpl_int_t n, armpl_int_t kl,
                                armpl_int_t ku, armpl_int_t nrhs, double *ab,
                                armpl_int_t ldab, double *afb,
                                armpl_int_t ldafb, armpl_int_t *ipiv,
                                char *equed, double *r, double *c, double *b,
                                armpl_int_t ldb, double *x, armpl_int_t ldx,
                                double *rcond, double *ferr, double *berr,
                                double *work, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbsvx_work*, *LAPACKE_dgbsvx_work*, *LAPACKE_sgbsvx_work* and *LAPACKE_zgbsvx_work*.

4.19.659 LAPACKE_dgbsvxx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgbsvxx(armpl_int_t matrix_layout, char fact, char trans,
                            armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                            armpl_int_t nrhs, double *ab, armpl_int_t ldab,
                            double *afb, armpl_int_t ldafb, armpl_int_t *ipiv,
                            char *equed, double *r, double *c, double *b,
                            armpl_int_t ldb, double *x, armpl_int_t ldx,
                            double *rcond, double *rpvgrw, double *berr,
                            armpl_int_t n_err_bnds, double *err_bnds_norm,
                            double *err_bnds_comp, armpl_int_t nparams,
                            double *params);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbsvxx*, *LAPACKE_dgbsvxx*, *LAPACKE_sgbsvxx* and *LAPACKE_zgbsvxx*. It also exists with a native Fortran interface as *dgbsvxx*.

4.19.660 LAPACKE_dgbsvxx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgbsvxx_work(armpl_int_t matrix_layout, char fact,
                                 char trans, armpl_int_t n, armpl_int_t kl,
                                 armpl_int_t ku, armpl_int_t nrhs, double *ab,
                                 armpl_int_t ldab, double *afb,
                                 armpl_int_t ldafb, armpl_int_t *ipiv,
                                 char *equed, double *r, double *c, double *b,
                                 armpl_int_t ldb, double *x, armpl_int_t ldx,
                                 double *rcond, double *rpvgrw, double *berr,
                                 armpl_int_t n_err_bnds,
                                 double *err_bnds_norm, double *err_bnds_comp,
                                 armpl_int_t nparams, double *params,
                                 double *work, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbsvxx_work*, *LAPACKE_dgbsvxx_work*, *LAPACKE_sgbsvxx_work* and *LAPACKE_zgbsvxx_work*.

4.19.661 LAPACKE_dgbtrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgbtrf(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                           double *ab, armpl_int_t ldab, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbtrf*, *LAPACKE_dgbtrf*, *LAPACKE_sgbtrf* and *LAPACKE_zgbtrf*. It also exists with a native Fortran interface as *dgbtrf*.

4.19.662 LAPACKE_dgbtrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgbtrf_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                                double *ab, armpl_int_t ldab,
                                armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbtrf_work*, *LAPACKE_dgbtrf_work*, *LAPACKE_sgbtrf_work* and *LAPACKE_zgbtrf_work*.

4.19.663 LAPACKE_dgbtrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgbtrs(armpl_int_t matrix_layout, char trans,
                           armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                           armpl_int_t nrhs, const double *ab,
                           armpl_int_t ldab, const armpl_int_t *ipiv,
                           double *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbtrs*, *LAPACKE_dgbtrs*, *LAPACKE_sgbtrs* and *LAPACKE_zgbtrs*. It also exists with a native Fortran interface as *dgbtrs*.

4.19.664 LAPACKE_dgbtrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgbtrs_work(armpl_int_t matrix_layout, char trans,
                                armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                                armpl_int_t nrhs, const double *ab,
                                armpl_int_t ldab, const armpl_int_t *ipiv,
                                double *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbtrs_work*, *LAPACKE_dgbtrs_work*, *LAPACKE_sgbtrs_work* and *LAPACKE_zgbtrs_work*.

4.19.665 LAPACKE_dgebak

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgebak(armpl_int_t matrix_layout, char job, char side,
                           armpl_int_t n, armpl_int_t ilo, armpl_int_t ihi,
                           const double *scale, armpl_int_t m, double *v,
                           armpl_int_t ldv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgebak*, *LAPACKE_dgebak*, *LAPACKE_sgebak* and *LAPACKE_zgebak*. It also exists with a native Fortran interface as *dgebak*.

4.19.666 LAPACKE_dgebak_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgebak_work(armpl_int_t matrix_layout, char job,
                                char side, armpl_int_t n, armpl_int_t ilo,
                                armpl_int_t ihi, const double *scale,
                                armpl_int_t m, double *v, armpl_int_t ldv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgebak_work*, *LAPACKE_dgebak_work*, *LAPACKE_sgebak_work* and *LAPACKE_zgebak_work*.

4.19.667 LAPACKE_dgebal

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgebal(armpl_int_t matrix_layout, char job, armpl_int_t n,
                           double *a, armpl_int_t lda, armpl_int_t *ilo,
                           armpl_int_t *ihi, double *scale);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgebal*, *LAPACKE_dgebal*, *LAPACKE_sgebal* and *LAPACKE_zgebal*. It also exists with a native Fortran interface as *dgebal*.

4.19.668 LAPACKE_dgebal_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgebal_work(armpl_int_t matrix_layout, char job,
                                armpl_int_t n, double *a, armpl_int_t lda,
                                armpl_int_t *ilo, armpl_int_t *ihi,
                                double *scale);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgebal_work*, *LAPACKE_dgebal_work*, *LAPACKE_sgebal_work* and *LAPACKE_zgebal_work*.

4.19.669 LAPACKE_dgebrd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgebrd(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, double *a, armpl_int_t lda,
                           double *d, double *e, double *tauq, double *taup);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgebrd*, *LAPACKE_dgebrd*, *LAPACKE_sgebrd* and *LAPACKE_zgebrd*. It also exists with a native Fortran interface as *dgebrd*.

4.19.670 LAPACKE_dgebrd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgebrd_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, double *a, armpl_int_t lda,
                                double *d, double *e, double *tauq,
                                double *taup, double *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgebrd_work*, *LAPACKE_dgebrd_work*, *LAPACKE_sgebrd_work* and *LAPACKE_zgebrd_work*.

4.19.671 LAPACKE_dgecon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgecon(armpl_int_t matrix_layout, char norm,
                           armpl_int_t n, const double *a, armpl_int_t lda,
                           double anorm, double *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgecon*, *LAPACKE_dgecon*, *LAPACKE_sgecon* and *LAPACKE_zgecon*. It also exists with a native Fortran interface as *dgecon*.

4.19.672 LAPACKE_dgecon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgecon_work(armpl_int_t matrix_layout, char norm,
                                armpl_int_t n, const double *a,
                                armpl_int_t lda, double anorm, double *rcond,
                                double *work, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgecon_work*, *LAPACKE_dgecon_work*, *LAPACKE_sgecon_work* and *LAPACKE_zgecon_work*.

4.19.673 LAPACKE_dgeequ

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgeequ(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, const double *a, armpl_int_t lda,
                           double *r, double *c, double *rowcnd,
                           double *colcnd, double *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeequ*, *LAPACKE_dgeequ*, *LAPACKE_sgeequ* and *LAPACKE_zgeequ*. It also exists with a native Fortran interface as *dgeequ*.

4.19.674 LAPACKE_dgeequ_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgeequ_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, const double *a,
                                armpl_int_t lda, double *r, double *c,
                                double *rowcnd, double *colcnd,
                                double *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeequ_work*, *LAPACKE_dgeequ_work*, *LAPACKE_sgeequ_work* and *LAPACKE_zgeequ_work*.

4.19.675 LAPACKE_dgeequb

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgeequb(armpl_int_t matrix_layout, armpl_int_t m,
                            armpl_int_t n, const double *a, armpl_int_t lda,
                            double *r, double *c, double *rowcnd,
                            double *colcnd, double *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeequb*, *LAPACKE_dgeequb*, *LAPACKE_sgeequb* and *LAPACKE_zgeequb*. It also exists with a native Fortran interface as *dgeequb*.

4.19.676 LAPACKE_dgeequb_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgeequb_work(armpl_int_t matrix_layout, armpl_int_t m,
                                 armpl_int_t n, const double *a,
                                 armpl_int_t lda, double *r, double *c,
                                 double *rowcnd, double *colcnd,
                                 double *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeequb_work*, *LAPACKE_dgeequb_work*, *LAPACKE_sgeequb_work* and *LAPACKE_zgeequb_work*.

4.19.677 LAPACKE_dgees

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgees(armpl_int_t matrix_layout, char jobvs, char sort,
                          LAPACK_D_SELECT2 select, armpl_int_t n, double *a,
                          armpl_int_t lda, armpl_int_t *sdim, double *wr,
                          double *wi, double *vs, armpl_int_t ldvs);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgees*, *LAPACKE_dgees*, *LAPACKE_sgees* and *LAPACKE_zgees*. It also exists with a native Fortran interface as *dgees*.

4.19.678 LAPACKE_dgees_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgees_work(armpl_int_t matrix_layout, char jobvs,
                               char sort, LAPACK_D_SELECT2 select,
                               armpl_int_t n, double *a, armpl_int_t lda,
                               armpl_int_t *sdim, double *wr, double *wi,
                               double *vs, armpl_int_t ldvs, double *work,
                               armpl_int_t lwork, armpl_int_t *bwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgees_work*, *LAPACKE_dgees_work*, *LAPACKE_sgees_work* and *LAPACKE_zgees_work*.

4.19.679 LAPACKE_dgeesx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgeesx(armpl_int_t matrix_layout, char jobvs, char sort,
                           LAPACK_D_SELECT2 select, char sense, armpl_int_t n,
                           double *a, armpl_int_t lda, armpl_int_t *sdim,
```

(continues on next page)

(continued from previous page)

```
                                double *wr, double *wi, double *vs,
                                armpl_int_t ldvs, double *rconde, double *rcondv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeesx*, *LAPACKE_dgeesx*, *LAPACKE_sgeesx* and *LAPACKE_zgeesx*. It also exists with a native Fortran interface as *dgeesx*.

4.19.680 LAPACKE_dgeesx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgeesx_work(armpl_int_t matrix_layout, char jobvs,
                                char sort, LAPACK_D_SELECT2 select,
                                char sense, armpl_int_t n, double *a,
                                armpl_int_t lda, armpl_int_t *sdim,
                                double *wr, double *wi, double *vs,
                                armpl_int_t ldvs, double *rconde,
                                double *rcondv, double *work,
                                armpl_int_t lwork, armpl_int_t *iwork,
                                armpl_int_t liwork, armpl_int_t *bwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeesx_work*, *LAPACKE_dgeesx_work*, *LAPACKE_sgeesx_work* and *LAPACKE_zgeesx_work*.

4.19.681 LAPACKE_dgeev

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgeev(armpl_int_t matrix_layout, char jobvl, char jobvr,
                          armpl_int_t n, double *a, armpl_int_t lda,
                          double *wr, double *wi, double *vl,
                          armpl_int_t ldvl, double *vr, armpl_int_t ldvr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeev*, *LAPACKE_dgeev*, *LAPACKE_sgeev* and *LAPACKE_zgeev*. It also exists with a native Fortran interface as *dgeev*.

4.19.682 LAPACKE_dgeev_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgeev_work(armpl_int_t matrix_layout, char jobvl,
                               char jobvr, armpl_int_t n, double *a,
                               armpl_int_t lda, double *wr, double *wi,
                               double *vl, armpl_int_t ldvl, double *vr,
                               armpl_int_t ldvr, double *work,
                               armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeev_work*, *LAPACKE_dgeev_work*, *LAPACKE_sgeev_work* and *LAPACKE_zgeev_work*.

4.19.683 LAPACKE_dgeevx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgeevx(armpl_int_t matrix_layout, char balanc, char jobvl,
                           char jobvr, char sense, armpl_int_t n, double *a,
                           armpl_int_t lda, double *wr, double *wi,
                           double *vl, armpl_int_t ldvl, double *vr,
                           armpl_int_t ldvr, armpl_int_t *ilo,
                           armpl_int_t *ihi, double *scale, double *abnrm,
                           double *rconde, double *rcondv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeevx*, *LAPACKE_dgeevx*, *LAPACKE_sgeevx* and *LAPACKE_zgeevx*. It also exists with a native Fortran interface as *dgeevx*.

4.19.684 LAPACKE_dgeevx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgeevx_work(armpl_int_t matrix_layout, char balanc,
                                char jobvl, char jobvr, char sense,
                                armpl_int_t n, double *a, armpl_int_t lda,
                                double *wr, double *wi, double *vl,
                                armpl_int_t ldvl, double *vr,
                                armpl_int_t ldvr, armpl_int_t *ilo,
                                armpl_int_t *ihi, double *scale,
                                double *abnrm, double *rconde, double *rcondv,
                                double *work, armpl_int_t lwork,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeevx_work*, *LAPACKE_dgeevx_work*, *LAPACKE_sgeevx_work* and *LAPACKE_zgeevx_work*.

4.19.685 LAPACKE_dgehrd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgehrd(armpl_int_t matrix_layout, armpl_int_t n,
                           armpl_int_t ilo, armpl_int_t ihi, double *a,
                           armpl_int_t lda, double *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.686 LAPACKE_dgehrd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgehrd_work(armpl_int_t matrix_layout, armpl_int_t n,
                                armpl_int_t ilo, armpl_int_t ihi, double *a,
                                armpl_int_t lda, double *tau, double *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.687 LAPACKE_dgejsv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgejsv(armpl_int_t matrix_layout, char joba, char jobu,
                           char jobv, char jobr, char jobt, char jobp,
                           armpl_int_t m, armpl_int_t n, double *a,
                           armpl_int_t lda, double *sva, double *u,
                           armpl_int_t ldu, double *v, armpl_int_t ldv,
                           double *stat, armpl_int_t *istat);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgejsv*, *LAPACKE_dgejsv*, *LAPACKE_sgejsv* and *LAPACKE_zgejsv*. It also exists with a native Fortran interface as *dgejsv*.

4.19.688 LAPACKE_dgejsv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgejsv_work(armpl_int_t matrix_layout, char joba,
                                char jobu, char jobv, char jobr, char jobt,
                                char jobp, armpl_int_t m, armpl_int_t n,
                                double *a, armpl_int_t lda, double *sva,
                                double *u, armpl_int_t ldu, double *v,
                                armpl_int_t ldv, double *work,
                                armpl_int_t lwork, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgejsv_work*, *LAPACKE_dgejsv_work*, *LAPACKE_sgejsv_work* and *LAPACKE_zgejsv_work*.

4.19.689 LAPACKE_dgelq

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgelq(armpl_int_t matrix_layout, armpl_int_t m,
                          armpl_int_t n, double *a, armpl_int_t lda,
                          double *t, armpl_int_t tsize);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelq*, *LAPACKE_dgelq*, *LAPACKE_sgelq* and *LAPACKE_zgelq*. It also exists with a native Fortran interface as *dgelq*.

4.19.690 LAPACKE_dgelq2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgelq2(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, double *a, armpl_int_t lda,
                           double *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelq2*, *LAPACKE_dgelq2*, *LAPACKE_sgelq2* and *LAPACKE_zgelq2*. It also exists with a native Fortran interface as *dgelq2*.

4.19.691 LAPACKE_dgelq2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgelq2_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, double *a, armpl_int_t lda,
                                double *tau, double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelq2_work*, *LAPACKE_dgelq2_work*, *LAPACKE_sgelq2_work* and *LAPACKE_zgelq2_work*.

4.19.692 LAPACKE_dgelq_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgelq_work(armpl_int_t matrix_layout, armpl_int_t m,
                               armpl_int_t n, double *a, armpl_int_t lda,
                               double *t, armpl_int_t tsize, double *work,
                               armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelq_work*, *LAPACKE_dgelq_work*, *LAPACKE_sgelq_work* and *LAPACKE_zgelq_work*.

4.19.693 LAPACKE_dgelqf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgelqf(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, double *a, armpl_int_t lda,
                           double *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelqf*, *LAPACKE_dgelqf*, *LAPACKE_sgelqf* and *LAPACKE_zgelqf*. It also exists with a native Fortran interface as *dgelqf*.

4.19.694 LAPACKE_dgelqf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgelqf_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, double *a, armpl_int_t lda,
                                double *tau, double *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelqf_work*, *LAPACKE_dgelqf_work*, *LAPACKE_sgelqf_work* and *LAPACKE_zgelqf_work*.

4.19.695 LAPACKE_dgels

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgels(armpl_int_t matrix_layout, char trans,
                          armpl_int_t m, armpl_int_t n, armpl_int_t nrhs,
                          double *a, armpl_int_t lda, double *b,
                          armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgels*, *LAPACKE_dgels*, *LAPACKE_sgels* and *LAPACKE_zgels*. It also exists with a native Fortran interface as *dgels*.

4.19.696 LAPACKE_dgels_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgels_work(armpl_int_t matrix_layout, char trans,
                               armpl_int_t m, armpl_int_t n, armpl_int_t nrhs,
                               double *a, armpl_int_t lda, double *b,
                               armpl_int_t ldb, double *work,
                               armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgels_work*, *LAPACKE_dgels_work*, *LAPACKE_sgels_work* and *LAPACKE_zgels_work*.

4.19.697 LAPACKE_dgelsd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgelsd(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t nrhs, double *a,
                           armpl_int_t lda, double *b, armpl_int_t ldb,
                           double *s, double rcond, armpl_int_t *rank);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelsd*, *LAPACKE_dgelsd*, *LAPACKE_sgelsd* and *LAPACKE_zgelsd*. It also exists with a native Fortran interface as *dgelsd*.

4.19.698 LAPACKE_dgelsd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgelsd_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t nrhs, double *a,
                                armpl_int_t lda, double *b, armpl_int_t ldb,
                                double *s, double rcond, armpl_int_t *rank,
                                double *work, armpl_int_t lwork,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelsd_work*, *LAPACKE_dgelsd_work*, *LAPACKE_sgelsd_work* and *LAPACKE_zgelsd_work*.

4.19.699 LAPACKE_dgelss

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgelss(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t nrhs, double *a,
                           armpl_int_t lda, double *b, armpl_int_t ldb,
                           double *s, double rcond, armpl_int_t *rank);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelss*, *LAPACKE_dgelss*, *LAPACKE_sgelss* and *LAPACKE_zgelss*. It also exists with a native Fortran interface as *dgelss*.

4.19.700 LAPACKE_dgelss_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgelss_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t nrhs, double *a,
                                armpl_int_t lda, double *b, armpl_int_t ldb,
                                double *s, double rcond, armpl_int_t *rank,
                                double *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelss_work*, *LAPACKE_dgelss_work*, *LAPACKE_sgelss_work* and *LAPACKE_zgelss_work*.

4.19.701 LAPACKE_dgelsy

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgelsy(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t nrhs, double *a,
                           armpl_int_t lda, double *b, armpl_int_t ldb,
                           armpl_int_t *jpvt, double rcond,
                           armpl_int_t *rank);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelsy*, *LAPACKE_dgelsy*, *LAPACKE_sgelsy* and *LAPACKE_zgelsy*. It also exists with a native Fortran interface as *dgelsy*.

4.19.702 LAPACKE_dgelsy_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgelsy_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t nrhs, double *a,
                                armpl_int_t lda, double *b, armpl_int_t ldb,
                                armpl_int_t *jpvt, double rcond,
                                armpl_int_t *rank, double *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelsy_work*, *LAPACKE_dgelsy_work*, *LAPACKE_sgelsy_work* and *LAPACKE_zgelsy_work*.

4.19.703 LAPACKE_dgemlq

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgemlq(armpl_int_t matrix_layout, char side, char trans,
                           armpl_int_t m, armpl_int_t n, armpl_int_t k,
                           const double *a, armpl_int_t lda, const double *t,
                           armpl_int_t tsize, double *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgemlq*, *LAPACKE_dgemlq*, *LAPACKE_sgemlq* and *LAPACKE_zgemlq*. It also exists with a native Fortran interface as *dgemlq*.

4.19.704 LAPACKE_dgemlq_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgemlq_work(armpl_int_t matrix_layout, char side,
                                char trans, armpl_int_t m, armpl_int_t n,
                                armpl_int_t k, const double *a,
                                armpl_int_t lda, const double *t,
                                armpl_int_t tsize, double *c, armpl_int_t ldc,
                                double *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgemlq_work*, *LAPACKE_dgemlq_work*, *LAPACKE_sgemlq_work* and *LAPACKE_zgemlq_work*.

4.19.705 LAPACKE_dgemqr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgemqr(armpl_int_t matrix_layout, char side, char trans,
                           armpl_int_t m, armpl_int_t n, armpl_int_t k,
                           const double *a, armpl_int_t lda, const double *t,
                           armpl_int_t tsize, double *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgemqr*, *LAPACKE_dgemqr*, *LAPACKE_sgemqr* and *LAPACKE_zgemqr*. It also exists with a native Fortran interface as *dgemqr*.

4.19.706 LAPACKE_dgemqr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgemqr_work(armpl_int_t matrix_layout, char side,
                                char trans, armpl_int_t m, armpl_int_t n,
                                armpl_int_t k, const double *a,
                                armpl_int_t lda, const double *t,
                                armpl_int_t tsize, double *c, armpl_int_t ldc,
                                double *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgemqr_work*, *LAPACKE_dgemqr_work*, *LAPACKE_sgemqr_work* and *LAPACKE_zgemqr_work*.

4.19.707 LAPACKE_dgemqrt

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgemqrt(armpl_int_t matrix_layout, char side, char trans,
                            armpl_int_t m, armpl_int_t n, armpl_int_t k,
                            armpl_int_t nb, const double *v, armpl_int_t ldv,
                            const double *t, armpl_int_t ldt, double *c,
                            armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgemqrt*, *LAPACKE_dgemqrt*, *LAPACKE_sgemqrt* and *LAPACKE_zgemqrt*. It also exists with a native Fortran interface as *dgemqrt*.

4.19.708 LAPACKE_dgemqrt_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgemqrt_work(armpl_int_t matrix_layout, char side,
                                 char trans, armpl_int_t m, armpl_int_t n,
                                 armpl_int_t k, armpl_int_t nb,
                                 const double *v, armpl_int_t ldv,
```

(continues on next page)

(continued from previous page)

```
                                const double *t, armpl_int_t ldt, double *c,
                                armpl_int_t ldc, double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgemqrt_work*, *LAPACKE_dgemqrt_work*, *LAPACKE_sgemqrt_work* and *LAPACKE_zgemqrt_work*.

4.19.709 LAPACKE_dgeqlf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgeqlf(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, double *a, armpl_int_t lda,
                           double *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqlf*, *LAPACKE_dgeqlf*, *LAPACKE_sgeqlf* and *LAPACKE_zgeqlf*. It also exists with a native Fortran interface as *dgeqlf*.

4.19.710 LAPACKE_dgeqlf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgeqlf_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, double *a, armpl_int_t lda,
                                double *tau, double *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqlf_work*, *LAPACKE_dgeqlf_work*, *LAPACKE_sgeqlf_work* and *LAPACKE_zgeqlf_work*.

4.19.711 LAPACKE_dgeqp3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgeqp3(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, double *a, armpl_int_t lda,
                           armpl_int_t *jpvt, double *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqp3*, *LAPACKE_dgeqp3*, *LAPACKE_sgeqp3* and *LAPACKE_zgeqp3*. It also exists with a native Fortran interface as *dgeqp3*.

4.19.712 LAPACKE_dgeqp3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgeqp3_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, double *a, armpl_int_t lda,
                                armpl_int_t *jpvt, double *tau, double *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqp3_work*, *LAPACKE_dgeqp3_work*, *LAPACKE_sgeqp3_work* and *LAPACKE_zgeqp3_work*.

4.19.713 LAPACKE_dgeqpf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgeqpf(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, double *a, armpl_int_t lda,
                           armpl_int_t *jpvt, double *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqpf*, *LAPACKE_dgeqpf*, *LAPACKE_sgeqpf* and *LAPACKE_zgeqpf*.

4.19.714 LAPACKE_dgeqpf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgeqpf_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, double *a, armpl_int_t lda,
                                armpl_int_t *jpvt, double *tau,
                                double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqpf_work*, *LAPACKE_dgeqpf_work*, *LAPACKE_sgeqpf_work* and *LAPACKE_zgeqpf_work*.

4.19.715 LAPACKE_dgeqr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgeqr(armpl_int_t matrix_layout, armpl_int_t m,
                          armpl_int_t n, double *a, armpl_int_t lda,
                          double *t, armpl_int_t tsize);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqr*, *LAPACKE_dgeqr*, *LAPACKE_sgeqr* and *LAPACKE_zgeqr*. It also exists with a native Fortran interface as *dgeqr*.

4.19.716 LAPACKE_dgeqr2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgeqr2(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, double *a, armpl_int_t lda,
                           double *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqr2*, *LAPACKE_dgeqr2*, *LAPACKE_sgeqr2* and *LAPACKE_zgeqr2*. It also exists with a native Fortran interface as *dgeqr2*.

4.19.717 LAPACKE_dgeqr2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgeqr2_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, double *a, armpl_int_t lda,
                                double *tau, double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqr2_work*, *LAPACKE_dgeqr2_work*, *LAPACKE_sgeqr2_work* and *LAPACKE_zgeqr2_work*.

4.19.718 LAPACKE_dgeqr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgeqr_work(armpl_int_t matrix_layout, armpl_int_t m,
                               armpl_int_t n, double *a, armpl_int_t lda,
                               double *t, armpl_int_t tsize, double *work,
                               armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqr_work*, *LAPACKE_dgeqr_work*, *LAPACKE_sgeqr_work* and *LAPACKE_zgeqr_work*.

4.19.719 LAPACKE_dgeqrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgeqrf(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, double *a, armpl_int_t lda,
                           double *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrf*, *LAPACKE_dgeqrf*, *LAPACKE_sgeqrf* and *LAPACKE_zgeqrf*. It also exists with a native Fortran interface as *dgeqrf*.

4.19.720 LAPACKE_dgeqrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgeqrf_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, double *a, armpl_int_t lda,
                                double *tau, double *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrf_work*, *LAPACKE_dgeqrf_work*, *LAPACKE_sgeqrf_work* and *LAPACKE_zgeqrf_work*.

4.19.721 LAPACKE_dgeqrfp

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgeqrfp(armpl_int_t matrix_layout, armpl_int_t m,
                            armpl_int_t n, double *a, armpl_int_t lda,
                            double *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrfp*, *LAPACKE_dgeqrfp*, *LAPACKE_sgeqrfp* and *LAPACKE_zgeqrfp*. It also exists with a native Fortran interface as *dgeqrfp*.

4.19.722 LAPACKE_dgeqrfp_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgeqrfp_work(armpl_int_t matrix_layout, armpl_int_t m,
                                 armpl_int_t n, double *a, armpl_int_t lda,
                                 double *tau, double *work,
                                 armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrfp_work*, *LAPACKE_dgeqrfp_work*, *LAPACKE_sgeqrfp_work* and *LAPACKE_zgeqrfp_work*.

4.19.723 LAPACKE_dgeqrt

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgeqrt(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t nb, double *a,
                           armpl_int_t lda, double *t, armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrt*, *LAPACKE_dgeqrt*, *LAPACKE_sgeqrt* and *LAPACKE_zgeqrt*. It also exists with a native Fortran interface as *dgeqrt*.

4.19.724 LAPACKE_dgeqrt2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgeqrt2(armpl_int_t matrix_layout, armpl_int_t m,
                            armpl_int_t n, double *a, armpl_int_t lda,
                            double *t, armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrt2*, *LAPACKE_dgeqrt2*, *LAPACKE_sgeqrt2* and *LAPACKE_zgeqrt2*. It also exists with a native Fortran interface as *dgeqrt2*.

4.19.725 LAPACKE_dgeqrt2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgeqrt2_work(armpl_int_t matrix_layout, armpl_int_t m,
                                 armpl_int_t n, double *a, armpl_int_t lda,
                                 double *t, armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrt2_work*, *LAPACKE_dgeqrt2_work*, *LAPACKE_sgeqrt2_work* and *LAPACKE_zgeqrt2_work*.

4.19.726 LAPACKE_dgeqrt3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgeqrt3(armpl_int_t matrix_layout, armpl_int_t m,
                            armpl_int_t n, double *a, armpl_int_t lda,
                            double *t, armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrt3*, *LAPACKE_dgeqrt3*, *LAPACKE_sgeqrt3* and *LAPACKE_zgeqrt3*. It also exists with a native Fortran interface as *dgeqrt3*.

4.19.727 LAPACKE_dgeqrt3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgeqrt3_work(armpl_int_t matrix_layout, armpl_int_t m,
                                 armpl_int_t n, double *a, armpl_int_t lda,
                                 double *t, armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrt3_work*, *LAPACKE_dgeqrt3_work*, *LAPACKE_sgeqrt3_work* and *LAPACKE_zgeqrt3_work*.

4.19.728 LAPACKE_dgeqrt_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgeqrt_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t nb, double *a,
                                armpl_int_t lda, double *t, armpl_int_t ldt,
                                double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrt_work*, *LAPACKE_dgeqrt_work*, *LAPACKE_sgeqrt_work* and *LAPACKE_zgeqrt_work*.

4.19.729 LAPACKE_dgerfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgerfs(armpl_int_t matrix_layout, char trans,
                           armpl_int_t n, armpl_int_t nrhs, const double *a,
                           armpl_int_t lda, const double *af,
                           armpl_int_t ldaf, const armpl_int_t *ipiv,
                           const double *b, armpl_int_t ldb, double *x,
                           armpl_int_t ldx, double *ferr, double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgerfs*, *LAPACKE_dgerfs*, *LAPACKE_sgerfs* and *LAPACKE_zgerfs*. It also exists with a native Fortran interface as *dgerfs*.

4.19.730 LAPACKE_dgerfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgerfs_work(armpl_int_t matrix_layout, char trans,
                                armpl_int_t n, armpl_int_t nrhs,
                                const double *a, armpl_int_t lda,
                                const double *af, armpl_int_t ldaf,
                                const armpl_int_t *ipiv, const double *b,
                                armpl_int_t ldb, double *x, armpl_int_t ldx,
                                double *ferr, double *berr, double *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgerfs_work*, *LAPACKE_dgerfs_work*, *LAPACKE_sgerfs_work* and *LAPACKE_zgerfs_work*.

4.19.731 LAPACKE_dgerfsx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgerfsx(armpl_int_t matrix_layout, char trans, char equed,
                            armpl_int_t n, armpl_int_t nrhs, const double *a,
                            armpl_int_t lda, const double *af,
                            armpl_int_t ldaf, const armpl_int_t *ipiv,
                            const double *r, const double *c, const double *b,
                            armpl_int_t ldb, double *x, armpl_int_t ldx,
                            double *rcond, double *berr,
                            armpl_int_t n_err_bnds, double *err_bnds_norm,
                            double *err_bnds_comp, armpl_int_t nparams,
                            double *params);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgerfsx*, *LAPACKE_dgerfsx*, *LAPACKE_sgerfsx* and *LAPACKE_zgerfsx*. It also exists with a native Fortran interface as *dgerfsx*.

4.19.732 LAPACKE_dgerfsx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgerfsx_work(armpl_int_t matrix_layout, char trans,
                                 char equed, armpl_int_t n, armpl_int_t nrhs,
                                 const double *a, armpl_int_t lda,
                                 const double *af, armpl_int_t ldaf,
                                 const armpl_int_t *ipiv, const double *r,
                                 const double *c, const double *b,
                                 armpl_int_t ldb, double *x, armpl_int_t ldx,
                                 double *rcond, double *berr,
                                 armpl_int_t n_err_bnds,
                                 double *err_bnds_norm, double *err_bnds_comp,
                                 armpl_int_t nparams, double *params,
                                 double *work, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgerfsx_work*, *LAPACKE_dgerfsx_work*, *LAPACKE_sgerfsx_work* and *LAPACKE_zgerfsx_work*.

4.19.733 LAPACKE_dgerqf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgerqf(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, double *a, armpl_int_t lda,
                           double *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgerqf*, *LAPACKE_dgerqf*, *LAPACKE_sgerqf* and *LAPACKE_zgerqf*. It also exists with a native Fortran interface as *dgerqf*.

4.19.734 LAPACKE_dgerqf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgerqf_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, double *a, armpl_int_t lda,
                                double *tau, double *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgerqf_work*, *LAPACKE_dgerqf_work*, *LAPACKE_sgerqf_work* and *LAPACKE_zgerqf_work*.

4.19.735 LAPACKE_dgesdd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgesdd(armpl_int_t matrix_layout, char jobz,
                           armpl_int_t m, armpl_int_t n, double *a,
                           armpl_int_t lda, double *s, double *u,
                           armpl_int_t ldu, double *vt, armpl_int_t ldvt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesdd*, *LAPACKE_dgesdd*, *LAPACKE_sgesdd* and *LAPACKE_zgesdd*. It also exists with a native Fortran interface as *dgesdd*.

4.19.736 LAPACKE_dgesdd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgesdd_work(armpl_int_t matrix_layout, char jobz,
                                armpl_int_t m, armpl_int_t n, double *a,
                                armpl_int_t lda, double *s, double *u,
                                armpl_int_t ldu, double *vt, armpl_int_t ldvt,
                                double *work, armpl_int_t lwork,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesdd_work*, *LAPACKE_dgesdd_work*, *LAPACKE_sgesdd_work* and *LAPACKE_zgesdd_work*.

4.19.737 LAPACKE_dgesv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgesv(armpl_int_t matrix_layout, armpl_int_t n,
                          armpl_int_t nrhs, double *a, armpl_int_t lda,
                          armpl_int_t *ipiv, double *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesv*, *LAPACKE_dgesv*, *LAPACKE_sgesv* and *LAPACKE_zgesv*. It also exists with a native Fortran interface as *dgesv*.

4.19.738 LAPACKE_dgesv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgesv_work(armpl_int_t matrix_layout, armpl_int_t n,
                               armpl_int_t nrhs, double *a, armpl_int_t lda,
                               armpl_int_t *ipiv, double *b,
                               armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesv_work*, *LAPACKE_dgesv_work*, *LAPACKE_sgesv_work* and *LAPACKE_zgesv_work*.

4.19.739 LAPACKE_dgesvd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgesvd(armpl_int_t matrix_layout, char jobu, char jobvt,
                           armpl_int_t m, armpl_int_t n, double *a,
                           armpl_int_t lda, double *s, double *u,
                           armpl_int_t ldu, double *vt, armpl_int_t ldvt,
                           double *superb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvd*, *LAPACKE_dgesvd*, *LAPACKE_sgesvd* and *LAPACKE_zgesvd*. It also exists with a native Fortran interface as *dgesvd*.

4.19.740 LAPACKE_dgesvd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgesvd_work(armpl_int_t matrix_layout, char jobu,
                                char jobvt, armpl_int_t m, armpl_int_t n,
                                double *a, armpl_int_t lda, double *s,
                                double *u, armpl_int_t ldu, double *vt,
                                armpl_int_t ldvt, double *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvd_work*, *LAPACKE_dgesvd_work*, *LAPACKE_sgesvd_work* and *LAPACKE_zgesvd_work*.

4.19.741 LAPACKE_dgesvdx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgesvdx(armpl_int_t matrix_layout, char jobu, char jobvt,
                            char range, armpl_int_t m, armpl_int_t n,
                            double *a, armpl_int_t lda, double vl, double vu,
```

(continues on next page)

(continued from previous page)

```
                                armpl_int_t il, armpl_int_t iu, armpl_int_t *ns,
                                double *s, double *u, armpl_int_t ldu, double *vt,
                                armpl_int_t ldvt, armpl_int_t *superb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvdx*, *LAPACKE_dgesvdx*, *LAPACKE_sgesvdx* and *LAPACKE_zgesvdx*. It also exists with a native Fortran interface as *dgesvdx*.

4.19.742 LAPACKE_dgesvdx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgesvdx_work(armpl_int_t matrix_layout, char jobu,
                                 char jobvt, char range, armpl_int_t m,
                                 armpl_int_t n, double *a, armpl_int_t lda,
                                 double vl, double vu, armpl_int_t il,
                                 armpl_int_t iu, armpl_int_t *ns, double *s,
                                 double *u, armpl_int_t ldu, double *vt,
                                 armpl_int_t ldvt, double *work,
                                 armpl_int_t lwork, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvdx_work*, *LAPACKE_dgesvdx_work*, *LAPACKE_sgesvdx_work* and *LAPACKE_zgesvdx_work*.

4.19.743 LAPACKE_dgesvj

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgesvj(armpl_int_t matrix_layout, char joba, char jobu,
                           char jobv, armpl_int_t m, armpl_int_t n, double *a,
                           armpl_int_t lda, double *sva, armpl_int_t mv,
                           double *v, armpl_int_t ldv, double *stat);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvj*, *LAPACKE_dgesvj*, *LAPACKE_sgesvj* and *LAPACKE_zgesvj*. It also exists with a native Fortran interface as *dgesvj*.

4.19.744 LAPACKE_dgesvj_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgesvj_work(armpl_int_t matrix_layout, char joba,
                                char jobu, char jobv, armpl_int_t m,
                                armpl_int_t n, double *a, armpl_int_t lda,
                                double *sva, armpl_int_t mv, double *v,
                                armpl_int_t ldv, double *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.745 LAPACKE_dgesvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgesvx(armpl_int_t matrix_layout, char fact, char trans,
                           armpl_int_t n, armpl_int_t nrhs, double *a,
                           armpl_int_t lda, double *af, armpl_int_t ldaf,
                           armpl_int_t *ipiv, char *equed, double *r,
                           double *c, double *b, armpl_int_t ldb, double *x,
                           armpl_int_t ldx, double *rcond, double *ferr,
                           double *berr, double *rpivot);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvx*, *LAPACKE_dgesvx*, *LAPACKE_sgesvx* and *LAPACKE_zgesvx*. It also exists with a native Fortran interface as *dgesvx*.

4.19.746 LAPACKE_dgesvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgesvx_work(armpl_int_t matrix_layout, char fact,
                                char trans, armpl_int_t n, armpl_int_t nrhs,
                                double *a, armpl_int_t lda, double *af,
                                armpl_int_t ldaf, armpl_int_t *ipiv,
                                char *equed, double *r, double *c, double *b,
                                armpl_int_t ldb, double *x, armpl_int_t ldx,
                                double *rcond, double *ferr, double *berr,
                                double *work, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvx_work*, *LAPACKE_dgesvx_work*, *LAPACKE_sgesvx_work* and *LAPACKE_zgesvx_work*.

4.19.747 LAPACKE_dgesvxx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgesvxx(armpl_int_t matrix_layout, char fact, char trans,
                            armpl_int_t n, armpl_int_t nrhs, double *a,
                            armpl_int_t lda, double *af, armpl_int_t ldaf,
                            armpl_int_t *ipiv, char *equed, double *r,
                            double *c, double *b, armpl_int_t ldb, double *x,
                            armpl_int_t ldx, double *rcond, double *rpvgrw,
                            double *berr, armpl_int_t n_err_bnds,
                            double *err_bnds_norm, double *err_bnds_comp,
                            armpl_int_t nparams, double *params);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvxx*, *LAPACKE_dgesvxx*, *LAPACKE_sgesvxx* and *LAPACKE_zgesvxx*. It also exists with a native Fortran interface as *dgesvxx*.

4.19.748 LAPACKE_dgesvxx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgesvxx_work(armpl_int_t matrix_layout, char fact,
                                 char trans, armpl_int_t n, armpl_int_t nrhs,
                                 double *a, armpl_int_t lda, double *af,
                                 armpl_int_t ldaf, armpl_int_t *ipiv,
                                 char *equed, double *r, double *c, double *b,
                                 armpl_int_t ldb, double *x, armpl_int_t ldx,
                                 double *rcond, double *rpvgrw, double *berr,
```

(continues on next page)

(continued from previous page)

```
                                armpl_int_t n_err_bnds,
                                double *err_bnds_norm, double *err_bnds_comp,
                                armpl_int_t nparams, double *params,
                                double *work, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvxx_work*, *LAPACKE_dgesvxx_work*, *LAPACKE_sgesvxx_work* and *LAPACKE_zgesvxx_work*.

4.19.749 LAPACKE_dgetf2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgetf2(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, double *a, armpl_int_t lda,
                           armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetf2*, *LAPACKE_dgetf2*, *LAPACKE_sgetf2* and *LAPACKE_zgetf2*. It also exists with a native Fortran interface as *dgetf2*.

4.19.750 LAPACKE_dgetf2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgetf2_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, double *a, armpl_int_t lda,
                                armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetf2_work*, *LAPACKE_dgetf2_work*, *LAPACKE_sgetf2_work* and *LAPACKE_zgetf2_work*.

4.19.751 LAPACKE_dgetrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgetrf(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, double *a, armpl_int_t lda,
                           armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetrf*, *LAPACKE_dgetrf*, *LAPACKE_sgetrf* and *LAPACKE_zgetrf*. It also exists with a native Fortran interface as *dgetrf*.

4.19.752 LAPACKE_dgetrf2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgetrf2(armpl_int_t matrix_layout, armpl_int_t m,
                            armpl_int_t n, double *a, armpl_int_t lda,
                            armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.753 LAPACKE_dgetrf2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgetrf2_work(armpl_int_t matrix_layout, armpl_int_t m,
                                 armpl_int_t n, double *a, armpl_int_t lda,
                                 armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetrf2_work*, *LAPACKE_dgetrf2_work*, *LAPACKE_sgetrf2_work* and *LAPACKE_zgetrf2_work*.

4.19.754 LAPACKE_dgetrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgetrf_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, double *a, armpl_int_t lda,
                                armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetrf_work*, *LAPACKE_dgetrf_work*, *LAPACKE_sgetrf_work* and *LAPACKE_zgetrf_work*.

4.19.755 LAPACKE_dgetri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgetri(armpl_int_t matrix_layout, armpl_int_t n,
                           double *a, armpl_int_t lda,
                           const armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetri*, *LAPACKE_dgetri*, *LAPACKE_sgetri* and *LAPACKE_zgetri*. It also exists with a native Fortran interface as *dgetri*.

4.19.756 LAPACKE_dgetri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgetri_work(armpl_int_t matrix_layout, armpl_int_t n,
                                double *a, armpl_int_t lda,
                                const armpl_int_t *ipiv, double *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetri_work*, *LAPACKE_dgetri_work*, *LAPACKE_sgetri_work* and *LAPACKE_zgetri_work*.

4.19.757 LAPACKE_dgetrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgetrs(armpl_int_t matrix_layout, char trans,
                           armpl_int_t n, armpl_int_t nrhs, const double *a,
                           armpl_int_t lda, const armpl_int_t *ipiv,
                           double *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetrs*, *LAPACKE_dgetrs*, *LAPACKE_sgetrs* and *LAPACKE_zgetrs*. It also exists with a native Fortran interface as *dgetrs*.

4.19.758 LAPACKE_dgetrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgetrs_work(armpl_int_t matrix_layout, char trans,
                                armpl_int_t n, armpl_int_t nrhs,
                                const double *a, armpl_int_t lda,
                                const armpl_int_t *ipiv, double *b,
                                armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetrs_work*, *LAPACKE_dgetrs_work*, *LAPACKE_sgetrs_work* and *LAPACKE_zgetrs_work*.

4.19.759 LAPACKE_dgetsls

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgetsls(armpl_int_t matrix_layout, char trans,
                            armpl_int_t m, armpl_int_t n, armpl_int_t nrhs,
                            double *a, armpl_int_t lda, double *b,
                            armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetsls*, *LAPACKE_dgetsls*, *LAPACKE_sgetsls* and *LAPACKE_zgetsls*. It also exists with a native Fortran interface as *dgetsls*.

4.19.760 LAPACKE_dgetsls_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgetsls_work(armpl_int_t matrix_layout, char trans,
                                 armpl_int_t m, armpl_int_t n,
                                 armpl_int_t nrhs, double *a, armpl_int_t lda,
                                 double *b, armpl_int_t ldb, double *work,
                                 armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetsls_work*, *LAPACKE_dgetsls_work*, *LAPACKE_sgetsls_work* and *LAPACKE_zgetsls_work*.

4.19.761 LAPACKE_dggbak

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dggbak(armpl_int_t matrix_layout, char job, char side,
                           armpl_int_t n, armpl_int_t ilo, armpl_int_t ihi,
                           const double *lscale, const double *rscale,
                           armpl_int_t m, double *v, armpl_int_t ldv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggbak*, *LAPACKE_dggbak*, *LAPACKE_sggbak* and *LAPACKE_zggbak*. It also exists with a native Fortran interface as *dggbak*.

4.19.762 LAPACKE_dggbak_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dggbak_work(armpl_int_t matrix_layout, char job,
                                char side, armpl_int_t n, armpl_int_t ilo,
                                armpl_int_t ihi, const double *lscale,
                                const double *rscale, armpl_int_t m,
                                double *v, armpl_int_t ldv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggbak_work*, *LAPACKE_dggbak_work*, *LAPACKE_sggbak_work* and *LAPACKE_zggbak_work*.

4.19.763 LAPACKE_dggbal

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dggbal(armpl_int_t matrix_layout, char job, armpl_int_t n,
                           double *a, armpl_int_t lda, double *b,
                           armpl_int_t ldb, armpl_int_t *ilo,
                           armpl_int_t *ihi, double *lscale, double *rscale);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggbal*, *LAPACKE_dggbal*, *LAPACKE_sggbal* and *LAPACKE_zggbal*. It also exists with a native Fortran interface as *dggbal*.

4.19.764 LAPACKE_dggbal_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dggbal_work(armpl_int_t matrix_layout, char job,
                                armpl_int_t n, double *a, armpl_int_t lda,
                                double *b, armpl_int_t ldb, armpl_int_t *ilo,
                                armpl_int_t *ihi, double *lscale,
                                double *rscale, double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggbal_work*, *LAPACKE_dggbal_work*, *LAPACKE_sggbal_work* and *LAPACKE_zggbal_work*.

4.19.765 LAPACKE_dgges

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgges(armpl_int_t matrix_layout, char jobvsl, char jobvsr,
                          char sort, LAPACK_D_SELECT3 selctg, armpl_int_t n,
                          double *a, armpl_int_t lda, double *b,
                          armpl_int_t ldb, armpl_int_t *sdim, double *alphar,
                          double *alphai, double *beta, double *vsl,
                          armpl_int_t ldvsl, double *vsr, armpl_int_t ldvsr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgges*, *LAPACKE_dgges*, *LAPACKE_sgges* and *LAPACKE_zgges*. It also exists with a native Fortran interface as *dgges*.

4.19.766 LAPACKE_dgges3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgges3(armpl_int_t matrix_layout, char jobvsl,
                           char jobvsr, char sort, LAPACK_D_SELECT3 selctg,
                           armpl_int_t n, double *a, armpl_int_t lda,
                           double *b, armpl_int_t ldb, armpl_int_t *sdim,
                           double *alphar, double *alphai, double *beta,
                           double *vsl, armpl_int_t ldvsl, double *vsr,
                           armpl_int_t ldvsr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgges3*, *LAPACKE_dgges3*, *LAPACKE_sgges3* and *LAPACKE_zgges3*. It also exists with a native Fortran interface as *dgges3*.

4.19.767 LAPACKE_dgges3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgges3_work(armpl_int_t matrix_layout, char jobvsl,
                                char jobvsr, char sort,
                                LAPACK_D_SELECT3 selctg, armpl_int_t n,
                                double *a, armpl_int_t lda, double *b,
                                armpl_int_t ldb, armpl_int_t *sdim,
                                double *alphar, double *alphai, double *beta,
                                double *vsl, armpl_int_t ldvsl, double *vsr,
                                armpl_int_t ldvsr, double *work,
                                armpl_int_t lwork, armpl_int_t *bwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgges3_work*, *LAPACKE_dgges3_work*, *LAPACKE_sgges3_work* and *LAPACKE_zgges3_work*.

4.19.768 LAPACKE_dgges_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgges_work(armpl_int_t matrix_layout, char jobvsl,
                               char jobvsr, char sort,
                               LAPACK_D_SELECT3 selctg, armpl_int_t n,
                               double *a, armpl_int_t lda, double *b,
                               armpl_int_t ldb, armpl_int_t *sdim,
                               double *alphar, double *alphai, double *beta,
                               double *vsl, armpl_int_t ldvsl, double *vsr,
                               armpl_int_t ldvsr, double *work,
                               armpl_int_t lwork, armpl_int_t *bwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgges_work*, *LAPACKE_dgges_work*, *LAPACKE_sgges_work* and *LAPACKE_zgges_work*.

4.19.769 LAPACKE_dggesx

4.19.770 LAPACKE_dggesx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dggesx_work(armpl_int_t matrix_layout, char jobvsl,
                                char jobvsr, char sort,
                                LAPACK_D_SELECT3 selctg, char sense,
                                armpl_int_t n, double *a, armpl_int_t lda,
                                double *b, armpl_int_t ldb, armpl_int_t *sdim,
                                double *alphar, double *alphai, double *beta,
                                double *vsl, armpl_int_t ldvsl, double *vsr,
                                armpl_int_t ldvsr, double *rconde,
                                double *rcondv, double *work,
                                armpl_int_t lwork, armpl_int_t *iwork,
                                armpl_int_t liwork, armpl_int_t *bwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggesx_work*, *LAPACKE_dggesx_work*, *LAPACKE_sggesx_work* and *LAPACKE_zggesx_work*.

4.19.771 LAPACKE_dggev

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dggev(armpl_int_t matrix_layout, char jobvl, char jobvr,
                          armpl_int_t n, double *a, armpl_int_t lda,
                          double *b, armpl_int_t ldb, double *alphar,
                          double *alphai, double *beta, double *vl,
                          armpl_int_t ldvl, double *vr, armpl_int_t ldvr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggev*, *LAPACKE_dggev*, *LAPACKE_sggev* and *LAPACKE_zggev*. It also exists with a native Fortran interface as *dggev*.

4.19.772 LAPACKE_dggev3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dggev3(armpl_int_t matrix_layout, char jobvl, char jobvr,
                           armpl_int_t n, double *a, armpl_int_t lda,
                           double *b, armpl_int_t ldb, double *alphar,
                           double *alphai, double *beta, double *vl,
                           armpl_int_t ldvl, double *vr, armpl_int_t ldvr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggev3*, *LAPACKE_dggev3*, *LAPACKE_sggev3* and *LAPACKE_zggev3*. It also exists with a native Fortran interface as *dggev3*.

4.19.773 LAPACKE_dggev3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dggev3_work(armpl_int_t matrix_layout, char jobvl,
                                char jobvr, armpl_int_t n, double *a,
                                armpl_int_t lda, double *b, armpl_int_t ldb,
                                double *alphar, double *alphai, double *beta,
                                double *vl, armpl_int_t ldvl, double *vr,
                                armpl_int_t ldvr, double *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggev3_work*, *LAPACKE_dggev3_work*, *LAPACKE_sggev3_work* and *LAPACKE_zggev3_work*.

4.19.774 LAPACKE_dggev_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dggev_work(armpl_int_t matrix_layout, char jobvl,
                               char jobvr, armpl_int_t n, double *a,
                               armpl_int_t lda, double *b, armpl_int_t ldb,
                               double *alphar, double *alphai, double *beta,
                               double *vl, armpl_int_t ldvl, double *vr,
                               armpl_int_t ldvr, double *work,
                               armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggev_work*, *LAPACKE_dggev_work*, *LAPACKE_sggev_work* and *LAPACKE_zggev_work*.

4.19.775 LAPACKE_dggevx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dggevx(armpl_int_t matrix_layout, char balanc, char jobvl,
                           char jobvr, char sense, armpl_int_t n, double *a,
                           armpl_int_t lda, double *b, armpl_int_t ldb,
                           double *alphar, double *alphai, double *beta,
                           double *vl, armpl_int_t ldvl, double *vr,
                           armpl_int_t ldvr, armpl_int_t *ilo,
                           armpl_int_t *ihi, double *lscale, double *rscale,
                           double *abnrm, double *bbnrm, double *rconde,
                           double *rcondv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggevx*, *LAPACKE_dggevx*, *LAPACKE_sggevx* and *LAPACKE_zggevx*. It also exists with a native Fortran interface as *dggevx*.

4.19.776 LAPACKE_dggevx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dggevx_work(armpl_int_t matrix_layout, char balanc,
                                char jobvl, char jobvr, char sense,
                                armpl_int_t n, double *a, armpl_int_t lda,
                                double *b, armpl_int_t ldb, double *alphar,
                                double *alphai, double *beta, double *vl,
                                armpl_int_t ldvl, double *vr,
                                armpl_int_t ldvr, armpl_int_t *ilo,
                                armpl_int_t *ihi, double *lscale,
                                double *rscale, double *abnrm, double *bbnrm,
                                double *rconde, double *rcondv, double *work,
                                armpl_int_t lwork, armpl_int_t *iwork,
                                armpl_int_t *bwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggevx_work*, *LAPACKE_dggevx_work*, *LAPACKE_sggevx_work* and *LAPACKE_zggevx_work*.

4.19.777 LAPACKE_dggglm

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dggglm(armpl_int_t matrix_layout, armpl_int_t n,
                           armpl_int_t m, armpl_int_t p, double *a,
                           armpl_int_t lda, double *b, armpl_int_t ldb,
                           double *d, double *x, double *y);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggglm*, *LAPACKE_dggglm*, *LAPACKE_sggglm* and *LAPACKE_zggglm*. It also exists with a native Fortran interface as *dggglm*.

4.19.778 LAPACKE_dggglm_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dggglm_work(armpl_int_t matrix_layout, armpl_int_t n,
                                armpl_int_t m, armpl_int_t p, double *a,
                                armpl_int_t lda, double *b, armpl_int_t ldb,
                                double *d, double *x, double *y, double *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggglm_work*, *LAPACKE_dggglm_work*, *LAPACKE_sggglm_work* and *LAPACKE_zggglm_work*.

4.19.779 LAPACKE_dgghd3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgghd3(armpl_int_t matrix_layout, char compq, char compz,
                           armpl_int_t n, armpl_int_t ilo, armpl_int_t ihi,
                           double *a, armpl_int_t lda, double *b,
                           armpl_int_t ldb, double *q, armpl_int_t ldq,
                           double *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgghd3*, *LAPACKE_dgghd3*, *LAPACKE_sgghd3* and *LAPACKE_zgghd3*. It also exists with a native Fortran interface as *dgghd3*.

4.19.780 LAPACKE_dgghd3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgghd3_work(armpl_int_t matrix_layout, char compq,
                                char compz, armpl_int_t n, armpl_int_t ilo,
                                armpl_int_t ihi, double *a, armpl_int_t lda,
                                double *b, armpl_int_t ldb, double *q,
                                armpl_int_t ldq, double *z, armpl_int_t ldz,
                                double *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgghd3_work*, *LAPACKE_dgghd3_work*, *LAPACKE_sgghd3_work* and *LAPACKE_zgghd3_work*.

4.19.781 LAPACKE_dgghrd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgghrd(armpl_int_t matrix_layout, char compq, char compz,
                           armpl_int_t n, armpl_int_t ilo, armpl_int_t ihi,
                           double *a, armpl_int_t lda, double *b,
                           armpl_int_t ldb, double *q, armpl_int_t ldq,
                           double *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgghrd*, *LAPACKE_dgghrd*, *LAPACKE_sgghrd* and *LAPACKE_zgghrd*. It also exists with a native Fortran interface as *dgghrd*.

4.19.782 LAPACKE_dgghrd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgghrd_work(armpl_int_t matrix_layout, char compq,
                                char compz, armpl_int_t n, armpl_int_t ilo,
                                armpl_int_t ihi, double *a, armpl_int_t lda,
                                double *b, armpl_int_t ldb, double *q,
                                armpl_int_t ldq, double *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgghrd_work*, *LAPACKE_dgghrd_work*, *LAPACKE_sgghrd_work* and *LAPACKE_zgghrd_work*.

4.19.783 LAPACKE_dgglse

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgglse(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t p, double *a,
                           armpl_int_t lda, double *b, armpl_int_t ldb,
                           double *c, double *d, double *x);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgglse*, *LAPACKE_dgglse*, *LAPACKE_sgglse* and *LAPACKE_zgglse*. It also exists with a native Fortran interface as *dgglse*.

4.19.784 LAPACKE_dgglse_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgglse_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t p, double *a,
                                armpl_int_t lda, double *b, armpl_int_t ldb,
                                double *c, double *d, double *x, double *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgglse_work*, *LAPACKE_dgglse_work*, *LAPACKE_sgglse_work* and *LAPACKE_zgglse_work*.

4.19.785 LAPACKE_dggqrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dggqrf(armpl_int_t matrix_layout, armpl_int_t n,
                           armpl_int_t m, armpl_int_t p, double *a,
                           armpl_int_t lda, double *taua, double *b,
                           armpl_int_t ldb, double *taub);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggqrf*, *LAPACKE_dggqrf*, *LAPACKE_sggqrf* and *LAPACKE_zggqrf*. It also exists with a native Fortran interface as *dggqrf*.

4.19.786 LAPACKE_dggqrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dggqrf_work(armpl_int_t matrix_layout, armpl_int_t n,
                                armpl_int_t m, armpl_int_t p, double *a,
                                armpl_int_t lda, double *taua, double *b,
                                armpl_int_t ldb, double *taub, double *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggqrf_work*, *LAPACKE_dggqrf_work*, *LAPACKE_sggqrf_work* and *LAPACKE_zggqrf_work*.

4.19.787 LAPACKE_dggrqf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dggrqf(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t p, armpl_int_t n, double *a,
                           armpl_int_t lda, double *taua, double *b,
                           armpl_int_t ldb, double *taub);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggrqf*, *LAPACKE_dggrqf*, *LAPACKE_sggrqf* and *LAPACKE_zggrqf*. It also exists with a native Fortran interface as *dggrqf*.

4.19.788 LAPACKE_dggrqf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dggrqf_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t p, armpl_int_t n, double *a,
                                armpl_int_t lda, double *taua, double *b,
                                armpl_int_t ldb, double *taub, double *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggrqf_work*, *LAPACKE_dggrqf_work*, *LAPACKE_sggrqf_work* and *LAPACKE_zggrqf_work*.

4.19.789 LAPACKE_dggsvd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dggsvd(armpl_int_t matrix_layout, char jobu, char jobv,
                           char jobq, armpl_int_t m, armpl_int_t n,
                           armpl_int_t p, armpl_int_t *k, armpl_int_t *l,
                           double *a, armpl_int_t lda, double *b,
                           armpl_int_t ldb, double *alpha, double *beta,
                           double *u, armpl_int_t ldu, double *v,
                           armpl_int_t ldv, double *q, armpl_int_t ldq,
                           armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggsvd*, *LAPACKE_dggsvd*, *LAPACKE_sggsvd* and *LAPACKE_zggsvd*.

4.19.790 LAPACKE_dggsvd3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dggsvd3(armpl_int_t matrix_layout, char jobu, char jobv,
                            char jobq, armpl_int_t m, armpl_int_t n,
                            armpl_int_t p, armpl_int_t *k, armpl_int_t *l,
                            double *a, armpl_int_t lda, double *b,
                            armpl_int_t ldb, double *alpha, double *beta,
                            double *u, armpl_int_t ldu, double *v,
                            armpl_int_t ldv, double *q, armpl_int_t ldq,
                            armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggsvd3*, *LAPACKE_dggsvd3*, *LAPACKE_sggsvd3* and *LAPACKE_zggsvd3*. It also exists with a native Fortran interface as *dggsvd3*.

4.19.791 LAPACKE_dggsvd3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dggsvd3_work(armpl_int_t matrix_layout, char jobu,
                                 char jobv, char jobq, armpl_int_t m,
                                 armpl_int_t n, armpl_int_t p, armpl_int_t *k,
                                 armpl_int_t *l, double *a, armpl_int_t lda,
                                 double *b, armpl_int_t ldb, double *alpha,
                                 double *beta, double *u, armpl_int_t ldu,
                                 double *v, armpl_int_t ldv, double *q,
                                 armpl_int_t ldq, double *work,
                                 armpl_int_t lwork, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggsvd3_work*, *LAPACKE_dggsvd3_work*, *LAPACKE_sggsvd3_work* and *LAPACKE_zggsvd3_work*.

4.19.792 LAPACKE_dggsvd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dggsvd_work(armpl_int_t matrix_layout, char jobu,
                                char jobv, char jobq, armpl_int_t m,
                                armpl_int_t n, armpl_int_t p, armpl_int_t *k,
                                armpl_int_t *l, double *a, armpl_int_t lda,
                                double *b, armpl_int_t ldb, double *alpha,
                                double *beta, double *u, armpl_int_t ldu,
                                double *v, armpl_int_t ldv, double *q,
                                armpl_int_t ldq, double *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggsvd_work*, *LAPACKE_dggsvd_work*, *LAPACKE_sggsvd_work* and *LAPACKE_zggsvd_work*.

4.19.793 LAPACKE_dggsvp

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dggsvp(armpl_int_t matrix_layout, char jobu, char jobv,
                           char jobq, armpl_int_t m, armpl_int_t p,
                           armpl_int_t n, double *a, armpl_int_t lda,
                           double *b, armpl_int_t ldb, double tola,
                           double tolb, armpl_int_t *k, armpl_int_t *l,
                           double *u, armpl_int_t ldu, double *v,
                           armpl_int_t ldv, double *q, armpl_int_t ldq);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggsvp*, *LAPACKE_dggsvp*, *LAPACKE_sggsvp* and *LAPACKE_zggsvp*.

4.19.794 LAPACKE_dggsvp3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dggsvp3(armpl_int_t matrix_layout, char jobu, char jobv,
                            char jobq, armpl_int_t m, armpl_int_t p,
                            armpl_int_t n, double *a, armpl_int_t lda,
                            double *b, armpl_int_t ldb, double tola,
                            double tolb, armpl_int_t *k, armpl_int_t *l,
                            double *u, armpl_int_t ldu, double *v,
                            armpl_int_t ldv, double *q, armpl_int_t ldq);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggsvp3*, *LAPACKE_dggsvp3*, *LAPACKE_sggsvp3* and *LAPACKE_zggsvp3*. It also exists with a native Fortran interface as *dggsvp3*.

4.19.795 LAPACKE_dggsvp3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dggsvp3_work(armpl_int_t matrix_layout, char jobu,
                                 char jobv, char jobq, armpl_int_t m,
                                 armpl_int_t p, armpl_int_t n, double *a,
                                 armpl_int_t lda, double *b, armpl_int_t ldb,
                                 double tola, double tolb, armpl_int_t *k,
                                 armpl_int_t *l, double *u, armpl_int_t ldu,
                                 double *v, armpl_int_t ldv, double *q,
                                 armpl_int_t ldq, armpl_int_t *iwork,
                                 double *tau, double *work,
                                 armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggsvp3_work*, *LAPACKE_dggsvp3_work*, *LAPACKE_sggsvp3_work* and *LAPACKE_zggsvp3_work*.

4.19.796 LAPACKE_dggsvp_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dggsvp_work(armpl_int_t matrix_layout, char jobu,
                                char jobv, char jobq, armpl_int_t m,
                                armpl_int_t p, armpl_int_t n, double *a,
                                armpl_int_t lda, double *b, armpl_int_t ldb,
                                double tola, double tolb, armpl_int_t *k,
                                armpl_int_t *l, double *u, armpl_int_t ldu,
                                double *v, armpl_int_t ldv, double *q,
                                armpl_int_t ldq, armpl_int_t *iwork,
                                double *tau, double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggsvp_work*, *LAPACKE_dggsvp_work*, *LAPACKE_sggsvp_work* and *LAPACKE_zggsvp_work*.

4.19.797 LAPACKE_dgtcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgtcon(char norm, armpl_int_t n, const double *dl,
                           const double *d, const double *du,
                           const double *du2, const armpl_int_t *ipiv,
                           double anorm, double *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgtcon*, *LAPACKE_dgtcon*, *LAPACKE_sgtcon* and *LAPACKE_zgtcon*. It also exists with a native Fortran interface as *dgtcon*.

4.19.798 LAPACKE_dgtcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgtcon_work(char norm, armpl_int_t n, const double *dl,
                                const double *d, const double *du,
                                const double *du2, const armpl_int_t *ipiv,
                                double anorm, double *rcond, double *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgtcon_work*, *LAPACKE_dgtcon_work*, *LAPACKE_sgtcon_work* and *LAPACKE_zgtcon_work*.

4.19.799 LAPACKE_dgtrfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgtrfs(armpl_int_t matrix_layout, char trans,
                           armpl_int_t n, armpl_int_t nrhs, const double *dl,
                           const double *d, const double *du,
                           const double *dlf, const double *df,
                           const double *duf, const double *du2,
                           const armpl_int_t *ipiv, const double *b,
                           armpl_int_t ldb, double *x, armpl_int_t ldx,
                           double *ferr, double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgtrfs*, *LAPACKE_dgtrfs*, *LAPACKE_sgtrfs* and *LAPACKE_zgtrfs*. It also exists with a native Fortran interface as *dgtrfs*.

4.19.800 LAPACKE_dgtrfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgtrfs_work(armpl_int_t matrix_layout, char trans,
                                armpl_int_t n, armpl_int_t nrhs,
                                const double *dl, const double *d,
                                const double *du, const double *dlf,
                                const double *df, const double *duf,
                                const double *du2, const armpl_int_t *ipiv,
                                const double *b, armpl_int_t ldb, double *x,
                                armpl_int_t ldx, double *ferr, double *berr,
                                double *work, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgtrfs_work*, *LAPACKE_dgtrfs_work*, *LAPACKE_sgtrfs_work* and *LAPACKE_zgtrfs_work*.

4.19.801 LAPACKE_dgtsv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgtsv(armpl_int_t matrix_layout, armpl_int_t n,
                          armpl_int_t nrhs, double *dl, double *d, double *du,
                          double *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgtsv*, *LAPACKE_dgtsv*, *LAPACKE_sgtsv* and *LAPACKE_zgtsv*. It also exists with a native Fortran interface as *dgtsv*.

4.19.802 LAPACKE_dgtsv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgtsv_work(armpl_int_t matrix_layout, armpl_int_t n,
                               armpl_int_t nrhs, double *dl, double *d,
                               double *du, double *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgtsv_work*, *LAPACKE_dgtsv_work*, *LAPACKE_sgtsv_work* and *LAPACKE_zgtsv_work*.

4.19.803 LAPACKE_dgtsvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgtsvx(armpl_int_t matrix_layout, char fact, char trans,
                           armpl_int_t n, armpl_int_t nrhs, const double *dl,
                           const double *d, const double *du, double *dlf,
                           double *df, double *duf, double *du2,
                           armpl_int_t *ipiv, const double *b,
                           armpl_int_t ldb, double *x, armpl_int_t ldx,
                           double *rcond, double *ferr, double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgtsvx*, *LAPACKE_dgtsvx*, *LAPACKE_sgtsvx* and *LAPACKE_zgtsvx*. It also exists with a native Fortran interface as *dgtsvx*.

4.19.804 LAPACKE_dgtsvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgtsvx_work(armpl_int_t matrix_layout, char fact,
                                char trans, armpl_int_t n, armpl_int_t nrhs,
                                const double *dl, const double *d,
                                const double *du, double *dlf, double *df,
                                double *duf, double *du2, armpl_int_t *ipiv,
                                const double *b, armpl_int_t ldb, double *x,
                                armpl_int_t ldx, double *rcond, double *ferr,
                                double *berr, double *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgtsvx_work*, *LAPACKE_dgtsvx_work*, *LAPACKE_sgtsvx_work* and *LAPACKE_zgtsvx_work*.

4.19.805 LAPACKE_dgttrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgttrf(armpl_int_t n, double *dl, double *d, double *du,
                           double *du2, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgttrf*, *LAPACKE_dgttrf*, *LAPACKE_sgttrf* and *LAPACKE_zgttrf*. It also exists with a native Fortran interface as *dgttrf*.

4.19.806 LAPACKE_dgttrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgttrf_work(armpl_int_t n, double *dl, double *d,
                                double *du, double *du2, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgttrf_work*, *LAPACKE_dgttrf_work*, *LAPACKE_sgttrf_work* and *LAPACKE_zgttrf_work*.

4.19.807 LAPACKE_dgttrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgttrs(armpl_int_t matrix_layout, char trans,
                           armpl_int_t n, armpl_int_t nrhs, const double *dl,
                           const double *d, const double *du,
                           const double *du2, const armpl_int_t *ipiv,
                           double *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgttrs*, *LAPACKE_dgttrs*, *LAPACKE_sgttrs* and *LAPACKE_zgttrs*. It also exists with a native Fortran interface as *dgttrs*.

4.19.808 LAPACKE_dgttrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dgttrs_work(armpl_int_t matrix_layout, char trans,
                                armpl_int_t n, armpl_int_t nrhs,
                                const double *dl, const double *d,
                                const double *du, const double *du2,
```

(continues on next page)

(continued from previous page)

```
                                const armpl_int_t *ipiv, double *b,
                                armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgttrs_work*, *LAPACKE_dgttrs_work*, *LAPACKE_sgttrs_work* and *LAPACKE_zgttrs_work*.

4.19.809 LAPACKE_dhgeqz

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dhgeqz(armpl_int_t matrix_layout, char job, char compq,
                           char compz, armpl_int_t n, armpl_int_t ilo,
                           armpl_int_t ihi, double *h, armpl_int_t ldh,
                           double *t, armpl_int_t ldt, double *alphar,
                           double *alphai, double *beta, double *q,
                           armpl_int_t ldq, double *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chgeqz*, *LAPACKE_dhgeqz*, *LAPACKE_shgeqz* and *LAPACKE_zhgeqz*. It also exists with a native Fortran interface as *dhgeqz*.

4.19.810 LAPACKE_dhgeqz_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dhgeqz_work(armpl_int_t matrix_layout, char job,
                                char compq, char compz, armpl_int_t n,
                                armpl_int_t ilo, armpl_int_t ihi, double *h,
                                armpl_int_t ldh, double *t, armpl_int_t ldt,
                                double *alphar, double *alphai, double *beta,
                                double *q, armpl_int_t ldq, double *z,
                                armpl_int_t ldz, double *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chgeqz_work*, *LAPACKE_dhgeqz_work*, *LAPACKE_shgeqz_work* and *LAPACKE_zhgeqz_work*.

4.19.811 LAPACKE_dhsein

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dhsein(armpl_int_t matrix_layout, char job, char eigsrc,
                           char initv, armpl_int_t *select, armpl_int_t n,
                           const double *h, armpl_int_t ldh, double *wr,
                           const double *wi, double *vl, armpl_int_t ldvl,
                           double *vr, armpl_int_t ldvr, armpl_int_t mm,
                           armpl_int_t *m, armpl_int_t *ifaill,
                           armpl_int_t *ifailr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chsein*, *LAPACKE_dhsein*, *LAPACKE_shsein* and *LAPACKE_zhsein*. It also exists with a native Fortran interface as *dhsein*.

4.19.812 LAPACKE_dhsein_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dhsein_work(armpl_int_t matrix_layout, char job,
                                char eigsrc, char initv, armpl_int_t *select,
                                armpl_int_t n, const double *h,
                                armpl_int_t ldh, double *wr, const double *wi,
                                double *vl, armpl_int_t ldvl, double *vr,
                                armpl_int_t ldvr, armpl_int_t mm,
                                armpl_int_t *m, double *work,
                                armpl_int_t *ifaill, armpl_int_t *ifailr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chsein_work*, *LAPACKE_dhsein_work*, *LAPACKE_shsein_work* and *LAPACKE_zhsein_work*.

4.19.813 LAPACKE_dhseqr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dhseqr(armpl_int_t matrix_layout, char job, char compz,
                           armpl_int_t n, armpl_int_t ilo, armpl_int_t ihi,
                           double *h, armpl_int_t ldh, double *wr, double *wi,
                           double *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chseqr*, *LAPACKE_dhseqr*, *LAPACKE_shseqr* and *LAPACKE_zhseqr*. It also exists with a native Fortran interface as *dhseqr*.

4.19.814 LAPACKE_dhseqr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dhseqr_work(armpl_int_t matrix_layout, char job,
                                char compz, armpl_int_t n, armpl_int_t ilo,
                                armpl_int_t ihi, double *h, armpl_int_t ldh,
                                double *wr, double *wi, double *z,
                                armpl_int_t ldz, double *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chseqr_work*, *LAPACKE_dhseqr_work*, *LAPACKE_shseqr_work* and *LAPACKE_zhseqr_work*.

4.19.815 LAPACKE_dlacn2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dlacn2(armpl_int_t n, double *v, double *x,
                           armpl_int_t *isgn, double *est, armpl_int_t *kase,
                           armpl_int_t *isave);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clacn2*, *LAPACKE_dlacn2*, *LAPACKE_slacn2* and *LAPACKE_zlacn2*. It also exists with a native Fortran interface as *dlacn2*.

4.19.816 LAPACKE_dlacn2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dlacn2_work(armpl_int_t n, double *v, double *x,
                                armpl_int_t *isgn, double *est,
                                armpl_int_t *kase, armpl_int_t *isave);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clacn2_work*, *LAPACKE_dlacn2_work*, *LAPACKE_slacn2_work* and *LAPACKE_zlacn2_work*.

4.19.817 LAPACKE_dlacpy

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dlacpy(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t m, armpl_int_t n, const double *a,
                           armpl_int_t lda, double *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clacpy*, *LAPACKE_dlacpy*, *LAPACKE_slacpy* and *LAPACKE_zlacpy*. It also exists with a native Fortran interface as *dlacpy*.

4.19.818 LAPACKE_dlacpy_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dlacpy_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t m, armpl_int_t n, const double *a,
                                armpl_int_t lda, double *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clacpy_work*, *LAPACKE_dlacpy_work*, *LAPACKE_slacpy_work* and *LAPACKE_zlacpy_work*.

4.19.819 LAPACKE_dlag2s

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dlag2s(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, const double *a, armpl_int_t lda,
                           float *sa, armpl_int_t ldsa);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dlag2s*. It also exists with a native Fortran interface as *dlag2s*.

4.19.820 LAPACKE_dlag2s_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dlag2s_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, const double *a,
                                armpl_int_t lda, float *sa,
                                armpl_int_t ldsa);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dlag2s_work*.

4.19.821 LAPACKE_dlagge

Syntax

Routine is deprecated. Please choose a suitable alternative.

4.19.822 LAPACKE_dlagge_work

Syntax

Routine is deprecated. Please choose a suitable alternative.

4.19.823 LAPACKE_dlagsy

Syntax

Routine is deprecated. Please choose a suitable alternative.

4.19.824 LAPACKE_dlagsy_work

Syntax

Routine is deprecated. Please choose a suitable alternative.

4.19.825 LAPACKE_dlamch

Syntax

```
#include "armpl.h"

double LAPACKE_dlamch(char cmach);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dlamch* and *LAPACKE_slamch*.

4.19.826 LAPACKE_dlamch_work

Syntax

```
#include "armpl.h"

double LAPACKE_dlamch_work(char cmach);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dlamch_work* and *LAPACKE_slamch_work*.

4.19.827 LAPACKE_dlange

Syntax

```
#include "armpl.h"

double LAPACKE_dlange(armpl_int_t matrix_layout, char norm, armpl_int_t m,
                      armpl_int_t n, const double *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clange*, *LAPACKE_dlange*, *LAPACKE_slange* and *LAPACKE_zlange*. It also exists with a native Fortran interface as *dlange*.

4.19.828 LAPACKE_dlange_work

Syntax

```
#include "armpl.h"

double LAPACKE_dlange_work(armpl_int_t matrix_layout, char norm,
                           armpl_int_t m, armpl_int_t n, const double *a,
                           armpl_int_t lda, double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clange_work*, *LAPACKE_dlange_work*, *LAPACKE_slange_work* and *LAPACKE_zlange_work*.

4.19.829 LAPACKE_dlansy

Syntax

```
#include "armpl.h"

double LAPACKE_dlansy(armpl_int_t matrix_layout, char norm, char uplo,
                      armpl_int_t n, const double *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clansy*, *LAPACKE_dlansy*, *LAPACKE_slansy* and *LAPACKE_zlansy*. It also exists with a native Fortran interface as *dlansy*.

4.19.830 LAPACKE_dlansy_work

Syntax

```
#include "armpl.h"

double LAPACKE_dlansy_work(armpl_int_t matrix_layout, char norm, char uplo,
                           armpl_int_t n, const double *a, armpl_int_t lda,
                           double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clansy_work*, *LAPACKE_dlansy_work*, *LAPACKE_slansy_work* and *LAPACKE_zlansy_work*.

4.19.831 LAPACKE_dlantr

Syntax

```
#include "armpl.h"

double LAPACKE_dlantr(armpl_int_t matrix_layout, char norm, char uplo,
                      char diag, armpl_int_t m, armpl_int_t n,
                      const double *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clantr*, *LAPACKE_dlantr*, *LAPACKE_slantr* and *LAPACKE_zlantr*. It also exists with a native Fortran interface as *dlantr*.

4.19.832 LAPACKE_dlantr_work

Syntax

```
#include "armpl.h"

double LAPACKE_dlantr_work(armpl_int_t matrix_layout, char norm, char uplo,
                           char diag, armpl_int_t m, armpl_int_t n,
                           const double *a, armpl_int_t lda, double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clantr_work*, *LAPACKE_dlantr_work*, *LAPACKE_slantr_work* and *LAPACKE_zlantr_work*.

4.19.833 LAPACKE_dlapmr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dlapmr(armpl_int_t matrix_layout, armpl_int_t forwrd,
                           armpl_int_t m, armpl_int_t n, double *x,
                           armpl_int_t ldx, armpl_int_t *k);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clapmr*, *LAPACKE_dlapmr*, *LAPACKE_slapmr* and *LAPACKE_zlapmr*. It also exists with a native Fortran interface as *dlapmr*.

4.19.834 LAPACKE_dlapmr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dlapmr_work(armpl_int_t matrix_layout, armpl_int_t forwrd,
                                armpl_int_t m, armpl_int_t n, double *x,
                                armpl_int_t ldx, armpl_int_t *k);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clapmr_work*, *LAPACKE_dlapmr_work*, *LAPACKE_slapmr_work* and *LAPACKE_zlapmr_work*.

4.19.835 LAPACKE_dlapmt

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dlapmt(armpl_int_t matrix_layout, armpl_int_t forwrd,
                           armpl_int_t m, armpl_int_t n, double *x,
                           armpl_int_t ldx, armpl_int_t *k);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clapmt*, *LAPACKE_dlapmt*, *LAPACKE_slapmt* and *LAPACKE_zlapmt*. It also exists with a native Fortran interface as *dlapmt*.

4.19.836 LAPACKE_dlapmt_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dlapmt_work(armpl_int_t matrix_layout, armpl_int_t forwrd,
                                armpl_int_t m, armpl_int_t n, double *x,
                                armpl_int_t ldx, armpl_int_t *k);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clapmt_work*, *LAPACKE_dlapmt_work*, *LAPACKE_slapmt_work* and *LAPACKE_zlapmt_work*.

4.19.837 LAPACKE_dlapy2

Syntax

```
#include "armpl.h"

double LAPACKE_dlapy2(double x, double y);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dlapy2* and *LAPACKE_slapy2*. It also exists with a native Fortran interface as *dlapy2*.

4.19.838 LAPACKE_dlapy2_work

Syntax

```
#include "armpl.h"

double LAPACKE_dlapy2_work(double x, double y);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dlapy2_work* and *LAPACKE_slapy2_work*.

4.19.839 LAPACKE_dlapy3

Syntax

```
#include "armpl.h"

double LAPACKE_dlapy3(double x, double y, double z);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dlapy3* and *LAPACKE_slapy3*. It also exists with a native Fortran interface as *dlapy3*.

4.19.840 LAPACKE_dlapy3_work

Syntax

```
#include "armpl.h"

double LAPACKE_dlapy3_work(double x, double y, double z);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dlapy3_work* and *LAPACKE_slapy3_work*.

4.19.841 LAPACKE_dlarfb

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dlarfb(armpl_int_t matrix_layout, char side, char trans,
                           char direct, char storev, armpl_int_t m,
                           armpl_int_t n, armpl_int_t k, const double *v,
```

(continues on next page)

(continued from previous page)

```
                                armpl_int_t ldv, const double *t, armpl_int_t ldt,
                                double *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarfb*, *LAPACKE_dlarfb*, *LAPACKE_slarfb* and *LAPACKE_zlarfb*. It also exists with a native Fortran interface as *dlarfb*.

4.19.842 LAPACKE_dlarfb_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dlarfb_work(armpl_int_t matrix_layout, char side,
                                char trans, char direct, char storev,
                                armpl_int_t m, armpl_int_t n, armpl_int_t k,
                                const double *v, armpl_int_t ldv,
                                const double *t, armpl_int_t ldt, double *c,
                                armpl_int_t ldc, double *work,
                                armpl_int_t ldwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarfb_work*, *LAPACKE_dlarfb_work*, *LAPACKE_slarfb_work* and *LAPACKE_zlarfb_work*.

4.19.843 LAPACKE_dlarfg

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dlarfg(armpl_int_t n, double *alpha, double *x,
                           armpl_int_t incx, double *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarfg*, *LAPACKE_dlarfg*, *LAPACKE_slarfg* and *LAPACKE_zlarfg*. It also exists with a native Fortran interface as *dlarfg*.

4.19.844 LAPACKE_dlarfg_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dlarfg_work(armpl_int_t n, double *alpha, double *x,
                                armpl_int_t incx, double *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarfg_work*, *LAPACKE_dlarfg_work*, *LAPACKE_slarfg_work* and *LAPACKE_zlarfg_work*.

4.19.845 LAPACKE_dlarft

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dlarft(armpl_int_t matrix_layout, char direct,
                           char storev, armpl_int_t n, armpl_int_t k,
                           const double *v, armpl_int_t ldv,
                           const double *tau, double *t, armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarft*, *LAPACKE_dlarft*, *LAPACKE_slarft* and *LAPACKE_zlarft*. It also exists with a native Fortran interface as *dlarft*.

4.19.846 LAPACKE_dlarft_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dlarft_work(armpl_int_t matrix_layout, char direct,
                                char storev, armpl_int_t n, armpl_int_t k,
                                const double *v, armpl_int_t ldv,
                                const double *tau, double *t,
                                armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarft_work*, *LAPACKE_dlarft_work*, *LAPACKE_slarft_work* and *LAPACKE_zlarft_work*.

4.19.847 LAPACKE_dlarfx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dlarfx(armpl_int_t matrix_layout, char side,
                           armpl_int_t m, armpl_int_t n, const double *v,
                           double tau, double *c, armpl_int_t ldc,
                           double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarfx*, *LAPACKE_dlarfx*, *LAPACKE_slarfx* and *LAPACKE_zlarfx*. It also exists with a native Fortran interface as *dlarfx*.

4.19.848 LAPACKE_dlarfx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dlarfx_work(armpl_int_t matrix_layout, char side,
                                armpl_int_t m, armpl_int_t n, const double *v,
                                double tau, double *c, armpl_int_t ldc,
                                double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarfx_work*, *LAPACKE_dlarfx_work*, *LAPACKE_slarfx_work* and *LAPACKE_zlarfx_work*.

4.19.849 LAPACKE_dlarnv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dlarnv(armpl_int_t idist, armpl_int_t *iseed,
                           armpl_int_t n, double *x);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarnv*, *LAPACKE_dlarnv*, *LAPACKE_slarnv* and *LAPACKE_zlarnv*. It also exists with a native Fortran interface as *dlarnv*.

4.19.850 LAPACKE_dlarnv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dlarnv_work(armpl_int_t idist, armpl_int_t *iseed,
                                armpl_int_t n, double *x);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarnv_work*, *LAPACKE_dlarnv_work*, *LAPACKE_slarnv_work* and *LAPACKE_zlarnv_work*.

4.19.851 LAPACKE_dlartgp

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dlartgp(double f, double g, double *cs, double *sn,
                            double *r);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dlartgp* and *LAPACKE_slartgp*. It also exists with a native Fortran interface as *dlartgp*.

4.19.852 LAPACKE_dlartgp_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dlartgp_work(double f, double g, double *cs, double *sn,
                                 double *r);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dlartgp_work* and *LAPACKE_slartgp_work*.

4.19.853 LAPACKE_dlartgs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dlartgs(double x, double y, double sigma, double *cs,
                            double *sn);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dlartgs* and *LAPACKE_slartgs*. It also exists with a native Fortran interface as *dlartgs*.

4.19.854 LAPACKE_dlartgs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dlartgs_work(double x, double y, double sigma, double *cs,
                                 double *sn);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dlartgs_work* and *LAPACKE_slartgs_work*.

4.19.855 LAPACKE_dlascl

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dlascl(armpl_int_t matrix_layout, char type,
                           armpl_int_t kl, armpl_int_t ku, double cfrom,
                           double cto, armpl_int_t m, armpl_int_t n,
                           double *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clascl*, *LAPACKE_dlascl*, *LAPACKE_slascl* and *LAPACKE_zlascl*. It also exists with a native Fortran interface as *dlascl*.

4.19.856 LAPACKE_dlascl_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dlascl_work(armpl_int_t matrix_layout, char type,
                                armpl_int_t kl, armpl_int_t ku, double cfrom,
                                double cto, armpl_int_t m, armpl_int_t n,
                                double *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clascl_work*, *LAPACKE_dlascl_work*, *LAPACKE_slascl_work* and *LAPACKE_zlascl_work*.

4.19.857 LAPACKE_dlaset

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dlaset(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t m, armpl_int_t n, double alpha,
                           double beta, double *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_claset*, *LAPACKE_dlaset*, *LAPACKE_slaset* and *LAPACKE_zlaset*. It also exists with a native Fortran interface as *dlaset*.

4.19.858 LAPACKE_dlaset_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dlaset_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t m, armpl_int_t n, double alpha,
                                double beta, double *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_claset_work*, *LAPACKE_dlaset_work*, *LAPACKE_slaset_work* and *LAPACKE_zlaset_work*.

4.19.859 LAPACKE_dlasrt

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dlasrt(char id, armpl_int_t n, double *d);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dlasrt* and *LAPACKE_slasrt*. It also exists with a native Fortran interface as *dlasrt*.

4.19.860 LAPACKE_dlasrt_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dlasrt_work(char id, armpl_int_t n, double *d);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dlasrt_work* and *LAPACKE_slasrt_work*.

4.19.861 LAPACKE_dlassq

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dlassq(armpl_int_t n, double *x, armpl_int_t incx,
                           double *scale, double *sumsq);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_classq*, *LAPACKE_dlassq*, *LAPACKE_slassq* and *LAPACKE_zlassq*. It also exists with a native Fortran interface as *dlassq*.

4.19.862 LAPACKE_dlassq_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dlassq_work(armpl_int_t n, double *x, armpl_int_t incx,
                                double *scale, double *sumsq);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_classq_work*, *LAPACKE_dlassq_work*, *LAPACKE_slassq_work* and *LAPACKE_zlassq_work*.

4.19.863 LAPACKE_dlaswp

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dlaswp(armpl_int_t matrix_layout, armpl_int_t n,
                           double *a, armpl_int_t lda, armpl_int_t k1,
                           armpl_int_t k2, const armpl_int_t *ipiv,
                           armpl_int_t incx);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_claswp*, *LAPACKE_dlaswp*, *LAPACKE_slaswp* and *LAPACKE_zlaswp*. It also exists with a native Fortran interface as *dlaswp*.

4.19.864 LAPACKE_dlaswp_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dlaswp_work(armpl_int_t matrix_layout, armpl_int_t n,
                                double *a, armpl_int_t lda, armpl_int_t k1,
                                armpl_int_t k2, const armpl_int_t *ipiv,
                                armpl_int_t incx);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_claswp_work*, *LAPACKE_dlaswp_work*, *LAPACKE_slaswp_work* and *LAPACKE_zlaswp_work*.

4.19.865 LAPACKE_dlatms

Syntax

Routine is deprecated. Please choose a suitable alternative.

4.19.866 LAPACKE_dlatms_work

Syntax

Routine is deprecated. Please choose a suitable alternative.

4.19.867 LAPACKE_dlauum

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dlauum(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, double *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clauum*, *LAPACKE_dlauum*, *LAPACKE_slauum* and *LAPACKE_zlauum*. It also exists with a native Fortran interface as *dlauum*.

4.19.868 LAPACKE_dlauum_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dlauum_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, double *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clauum_work*, *LAPACKE_dlauum_work*, *LAPACKE_slauum_work* and *LAPACKE_zlauum_work*.

4.19.869 LAPACKE_dopgtr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dopgtr(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, const double *ap, const double *tau,
                           double *q, armpl_int_t ldq);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dopgtr* and *LAPACKE_sopgtr*. It also exists with a native Fortran interface as *dopgtr*.

4.19.870 LAPACKE_dopgtr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dopgtr_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, const double *ap,
                                const double *tau, double *q, armpl_int_t ldq,
                                double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dopgtr_work* and *LAPACKE_sopgtr_work*.

4.19.871 LAPACKE_dopmtr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dopmtr(armpl_int_t matrix_layout, char side, char uplo,
                           char trans, armpl_int_t m, armpl_int_t n,
                           const double *ap, const double *tau, double *c,
                           armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dopmtr* and *LAPACKE_sopmtr*. It also exists with a native Fortran interface as *dopmtr*.

4.19.872 LAPACKE_dopmtr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dopmtr_work(armpl_int_t matrix_layout, char side,
                                char uplo, char trans, armpl_int_t m,
                                armpl_int_t n, const double *ap,
                                const double *tau, double *c, armpl_int_t ldc,
                                double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dopmtr_work* and *LAPACKE_sopmtr_work*.

4.19.873 LAPACKE_dorbdb

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dorbdb(armpl_int_t matrix_layout, char trans, char signs,
                           armpl_int_t m, armpl_int_t p, armpl_int_t q,
                           double *x11, armpl_int_t ldx11, double *x12,
                           armpl_int_t ldx12, double *x21, armpl_int_t ldx21,
                           double *x22, armpl_int_t ldx22, double *theta,
                           double *phi, double *taup1, double *taup2,
                           double *tauq1, double *tauq2);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorbdb* and *LAPACKE_sorbdb*. It also exists with a native Fortran interface as *dorbdb*.

4.19.874 LAPACKE_dorbdb_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dorbdb_work(armpl_int_t matrix_layout, char trans,
                                char signs, armpl_int_t m, armpl_int_t p,
                                armpl_int_t q, double *x11, armpl_int_t ldx11,
                                double *x12, armpl_int_t ldx12, double *x21,
                                armpl_int_t ldx21, double *x22,
                                armpl_int_t ldx22, double *theta, double *phi,
                                double *taup1, double *taup2, double *tauq1,
                                double *tauq2, double *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorbdb_work* and *LAPACKE_sorbdb_work*.

4.19.875 LAPACKE_dorcsd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dorcsd(armpl_int_t matrix_layout, char jobu1, char jobu2,
                           char jobv1t, char jobv2t, char trans, char signs,
                           armpl_int_t m, armpl_int_t p, armpl_int_t q,
                           double *x11, armpl_int_t ldx11, double *x12,
                           armpl_int_t ldx12, double *x21, armpl_int_t ldx21,
                           double *x22, armpl_int_t ldx22, double *theta,
                           double *u1, armpl_int_t ldu1, double *u2,
                           armpl_int_t ldu2, double *v1t, armpl_int_t ldv1t,
                           double *v2t, armpl_int_t ldv2t);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorcsd* and *LAPACKE_sorcsd*. It also exists with a native Fortran interface as *dorcsd*.

4.19.876 LAPACKE_dorcsd2by1

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dorcsd2by1(armpl_int_t matrix_layout, char jobu1,
                               char jobu2, char jobv1t, armpl_int_t m,
                               armpl_int_t p, armpl_int_t q, double *x11,
                               armpl_int_t ldx11, double *x21,
                               armpl_int_t ldx21, double *theta, double *u1,
                               armpl_int_t ldu1, double *u2, armpl_int_t ldu2,
                               double *v1t, armpl_int_t ldv1t);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorcsd2by1* and *LAPACKE_sorcsd2by1*. It also exists with a native Fortran interface as *dorcsd2by1*.

4.19.877 LAPACKE_dorcsd2by1_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dorcsd2by1_work(armpl_int_t matrix_layout, char jobu1,
                                    char jobu2, char jobv1t, armpl_int_t m,
                                    armpl_int_t p, armpl_int_t q, double *x11,
                                    armpl_int_t ldx11, double *x21,
                                    armpl_int_t ldx21, double *theta,
                                    double *u1, armpl_int_t ldu1, double *u2,
                                    armpl_int_t ldu2, double *v1t,
                                    armpl_int_t ldv1t, double *work,
                                    armpl_int_t lwork, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorcsd2by1_work* and *LAPACKE_sorcsd2by1_work*.

4.19.878 LAPACKE_dorcsd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dorcsd_work(armpl_int_t matrix_layout, char jobu1,
                                char jobu2, char jobv1t, char jobv2t,
                                char trans, char signs, armpl_int_t m,
                                armpl_int_t p, armpl_int_t q, double *x11,
                                armpl_int_t ldx11, double *x12,
                                armpl_int_t ldx12, double *x21,
                                armpl_int_t ldx21, double *x22,
                                armpl_int_t ldx22, double *theta, double *u1,
                                armpl_int_t ldu1, double *u2,
                                armpl_int_t ldu2, double *v1t,
                                armpl_int_t ldv1t, double *v2t,
                                armpl_int_t ldv2t, double *work,
                                armpl_int_t lwork, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorcsd_work* and *LAPACKE_sorcsd_work*.

4.19.879 LAPACKE_dorgbr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dorgbr(armpl_int_t matrix_layout, char vect,
                           armpl_int_t m, armpl_int_t n, armpl_int_t k,
                           double *a, armpl_int_t lda, const double *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorgbr* and *LAPACKE_sorgbr*. It also exists with a native Fortran interface as *dorgbr*.

4.19.880 LAPACKE_dorgbr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dorgbr_work(armpl_int_t matrix_layout, char vect,
                                armpl_int_t m, armpl_int_t n, armpl_int_t k,
                                double *a, armpl_int_t lda, const double *tau,
                                double *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorgbr_work* and *LAPACKE_sorgbr_work*.

4.19.881 LAPACKE_dorghr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dorghr(armpl_int_t matrix_layout, armpl_int_t n,
                           armpl_int_t ilo, armpl_int_t ihi, double *a,
                           armpl_int_t lda, const double *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorghr* and *LAPACKE_sorghr*. It also exists with a native Fortran interface as *dorghr*.

4.19.882 LAPACKE_dorghr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dorghr_work(armpl_int_t matrix_layout, armpl_int_t n,
                                armpl_int_t ilo, armpl_int_t ihi, double *a,
                                armpl_int_t lda, const double *tau,
                                double *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorghr_work* and *LAPACKE_sorghr_work*.

4.19.883 LAPACKE_dorglq

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dorglq(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t k, double *a,
                           armpl_int_t lda, const double *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorglq* and *LAPACKE_sorglq*. It also exists with a native Fortran interface as *dorglq*.

4.19.884 LAPACKE_dorglq_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dorglq_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t k, double *a,
                                armpl_int_t lda, const double *tau,
                                double *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorglq_work* and *LAPACKE_sorglq_work*.

4.19.885 LAPACKE_dorgql

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dorgql(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t k, double *a,
                           armpl_int_t lda, const double *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorgql* and *LAPACKE_sorgql*. It also exists with a native Fortran interface as *dorgql*.

4.19.886 LAPACKE_dorgql_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dorgql_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t k, double *a,
                                armpl_int_t lda, const double *tau,
                                double *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorgql_work* and *LAPACKE_sorgql_work*.

4.19.887 LAPACKE_dorgqr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dorgqr(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t k, double *a,
                           armpl_int_t lda, const double *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorgqr* and *LAPACKE_sorgqr*. It also exists with a native Fortran interface as *dorgqr*.

4.19.888 LAPACKE_dorgqr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dorgqr_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t k, double *a,
                                armpl_int_t lda, const double *tau,
                                double *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorgqr_work* and *LAPACKE_sorgqr_work*.

4.19.889 LAPACKE_dorgrq

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dorgrq(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t k, double *a,
                           armpl_int_t lda, const double *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorgrq* and *LAPACKE_sorgrq*. It also exists with a native Fortran interface as *dorgrq*.

4.19.890 LAPACKE_dorgrq_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dorgrq_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t k, double *a,
                                armpl_int_t lda, const double *tau,
                                double *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorgrq_work* and *LAPACKE_sorgrq_work*.

4.19.891 LAPACKE_dorgtr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dorgtr(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, double *a, armpl_int_t lda,
                           const double *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorgtr* and *LAPACKE_sorgtr*. It also exists with a native Fortran interface as *dorgtr*.

4.19.892 LAPACKE_dorgtr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dorgtr_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, double *a, armpl_int_t lda,
                                const double *tau, double *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorgtr_work* and *LAPACKE_sorgtr_work*.

4.19.893 LAPACKE_dormbr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dormbr(armpl_int_t matrix_layout, char vect, char side,
                           char trans, armpl_int_t m, armpl_int_t n,
                           armpl_int_t k, const double *a, armpl_int_t lda,
                           const double *tau, double *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dormbr* and *LAPACKE_sormbr*. It also exists with a native Fortran interface as *dormbr*.

4.19.894 LAPACKE_dormbr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dormbr_work(armpl_int_t matrix_layout, char vect,
                                char side, char trans, armpl_int_t m,
                                armpl_int_t n, armpl_int_t k, const double *a,
                                armpl_int_t lda, const double *tau, double *c,
                                armpl_int_t ldc, double *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dormbr_work* and *LAPACKE_sormbr_work*.

4.19.895 LAPACKE_dormhr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dormhr(armpl_int_t matrix_layout, char side, char trans,
                           armpl_int_t m, armpl_int_t n, armpl_int_t ilo,
                           armpl_int_t ihi, const double *a, armpl_int_t lda,
                           const double *tau, double *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dormhr* and *LAPACKE_sormhr*. It also exists with a native Fortran interface as *dormhr*.

4.19.896 LAPACKE_dormhr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dormhr_work(armpl_int_t matrix_layout, char side,
                                char trans, armpl_int_t m, armpl_int_t n,
                                armpl_int_t ilo, armpl_int_t ihi,
                                const double *a, armpl_int_t lda,
                                const double *tau, double *c, armpl_int_t ldc,
                                double *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dormhr_work* and *LAPACKE_sormhr_work*.

4.19.897 LAPACKE_dormlq

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dormlq(armpl_int_t matrix_layout, char side, char trans,
                           armpl_int_t m, armpl_int_t n, armpl_int_t k,
                           const double *a, armpl_int_t lda,
                           const double *tau, double *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dormlq* and *LAPACKE_sormlq*. It also exists with a native Fortran interface as *dormlq*.

4.19.898 LAPACKE_dormlq_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dormlq_work(armpl_int_t matrix_layout, char side,
                                char trans, armpl_int_t m, armpl_int_t n,
                                armpl_int_t k, const double *a,
                                armpl_int_t lda, const double *tau, double *c,
                                armpl_int_t ldc, double *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dormlq_work* and *LAPACKE_sormlq_work*.

4.19.899 LAPACKE_dormql

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dormql(armpl_int_t matrix_layout, char side, char trans,
                           armpl_int_t m, armpl_int_t n, armpl_int_t k,
                           const double *a, armpl_int_t lda,
                           const double *tau, double *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dormql* and *LAPACKE_sormql*. It also exists with a native Fortran interface as *dormql*.

4.19.900 LAPACKE_dormql_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dormql_work(armpl_int_t matrix_layout, char side,
                                char trans, armpl_int_t m, armpl_int_t n,
                                armpl_int_t k, const double *a,
                                armpl_int_t lda, const double *tau, double *c,
                                armpl_int_t ldc, double *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dormql_work* and *LAPACKE_sormql_work*.

4.19.901 LAPACKE_dormqr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dormqr(armpl_int_t matrix_layout, char side, char trans,
                           armpl_int_t m, armpl_int_t n, armpl_int_t k,
                           const double *a, armpl_int_t lda,
                           const double *tau, double *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dormqr* and *LAPACKE_sormqr*. It also exists with a native Fortran interface as *dormqr*.

4.19.902 LAPACKE_dormqr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dormqr_work(armpl_int_t matrix_layout, char side,
                                char trans, armpl_int_t m, armpl_int_t n,
                                armpl_int_t k, const double *a,
                                armpl_int_t lda, const double *tau, double *c,
                                armpl_int_t ldc, double *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dormqr_work* and *LAPACKE_sormqr_work*.

4.19.903 LAPACKE_dormrq

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dormrq(armpl_int_t matrix_layout, char side, char trans,
                           armpl_int_t m, armpl_int_t n, armpl_int_t k,
                           const double *a, armpl_int_t lda,
                           const double *tau, double *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dormrq* and *LAPACKE_sormrq*. It also exists with a native Fortran interface as *dormrq*.

4.19.904 LAPACKE_dormrq_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dormrq_work(armpl_int_t matrix_layout, char side,
                                char trans, armpl_int_t m, armpl_int_t n,
                                armpl_int_t k, const double *a,
                                armpl_int_t lda, const double *tau, double *c,
                                armpl_int_t ldc, double *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dormrq_work* and *LAPACKE_sormrq_work*.

4.19.905 LAPACKE_dormrz

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dormrz(armpl_int_t matrix_layout, char side, char trans,
                           armpl_int_t m, armpl_int_t n, armpl_int_t k,
                           armpl_int_t l, const double *a, armpl_int_t lda,
                           const double *tau, double *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dormrz* and *LAPACKE_sormrz*. It also exists with a native Fortran interface as *dormrz*.

4.19.906 LAPACKE_dormrz_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dormrz_work(armpl_int_t matrix_layout, char side,
                                char trans, armpl_int_t m, armpl_int_t n,
                                armpl_int_t k, armpl_int_t l, const double *a,
                                armpl_int_t lda, const double *tau, double *c,
                                armpl_int_t ldc, double *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dormrz_work* and *LAPACKE_sormrz_work*.

4.19.907 LAPACKE_dormtr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dormtr(armpl_int_t matrix_layout, char side, char uplo,
                           char trans, armpl_int_t m, armpl_int_t n,
                           const double *a, armpl_int_t lda,
                           const double *tau, double *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dormtr* and *LAPACKE_sormtr*. It also exists with a native Fortran interface as *dormtr*.

4.19.908 LAPACKE_dormtr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dormtr_work(armpl_int_t matrix_layout, char side,
                                char uplo, char trans, armpl_int_t m,
                                armpl_int_t n, const double *a,
                                armpl_int_t lda, const double *tau, double *c,
                                armpl_int_t ldc, double *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dormtr_work* and *LAPACKE_sormtr_work*.

4.19.909 LAPACKE_dpbcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpbcon(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t kd, const double *ab,
                           armpl_int_t ldab, double anorm, double *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbcon*, *LAPACKE_dpbcon*, *LAPACKE_spbcon* and *LAPACKE_zpbcon*. It also exists with a native Fortran interface as *dpbcon*.

4.19.910 LAPACKE_dpbcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpbcon_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t kd,
                                const double *ab, armpl_int_t ldab,
                                double anorm, double *rcond, double *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbcon_work*, *LAPACKE_dpbcon_work*, *LAPACKE_spbcon_work* and *LAPACKE_zpbcon_work*.

4.19.911 LAPACKE_dpbequ

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpbequ(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t kd, const double *ab,
                           armpl_int_t ldab, double *s, double *scond,
                           double *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbequ*, *LAPACKE_dpbequ*, *LAPACKE_spbequ* and *LAPACKE_zpbequ*. It also exists with a native Fortran interface as *dpbequ*.

4.19.912 LAPACKE_dpbequ_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpbequ_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t kd,
                                const double *ab, armpl_int_t ldab, double *s,
                                double *scond, double *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbequ_work*, *LAPACKE_dpbequ_work*, *LAPACKE_spbequ_work* and *LAPACKE_zpbequ_work*.

4.19.913 LAPACKE_dpbrfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpbrfs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t kd, armpl_int_t nrhs,
                           const double *ab, armpl_int_t ldab,
                           const double *afb, armpl_int_t ldafb,
                           const double *b, armpl_int_t ldb, double *x,
                           armpl_int_t ldx, double *ferr, double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbrfs*, *LAPACKE_dpbrfs*, *LAPACKE_spbrfs* and *LAPACKE_zpbrfs*. It also exists with a native Fortran interface as *dpbrfs*.

4.19.914 LAPACKE_dpbrfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpbrfs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t kd,
                                armpl_int_t nrhs, const double *ab,
                                armpl_int_t ldab, const double *afb,
                                armpl_int_t ldafb, const double *b,
                                armpl_int_t ldb, double *x, armpl_int_t ldx,
                                double *ferr, double *berr, double *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbrfs_work*, *LAPACKE_dpbrfs_work*, *LAPACKE_spbrfs_work* and *LAPACKE_zpbrfs_work*.

4.19.915 LAPACKE_dpbstf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpbstf(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t kb, double *bb,
                           armpl_int_t ldbb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbstf*, *LAPACKE_dpbstf*, *LAPACKE_spbstf* and *LAPACKE_zpbstf*. It also exists with a native Fortran interface as *dpbstf*.

4.19.916 LAPACKE_dpbstf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpbstf_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t kb, double *bb,
                                armpl_int_t ldbb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.917 LAPACKE_dpbsv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpbsv(armpl_int_t matrix_layout, char uplo, armpl_int_t n,
                          armpl_int_t kd, armpl_int_t nrhs, double *ab,
                          armpl_int_t ldab, double *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbsv*, *LAPACKE_dpbsv*, *LAPACKE_spbsv* and *LAPACKE_zpbsv*. It also exists with a native Fortran interface as *dpbsv*.

4.19.918 LAPACKE_dpbsv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpbsv_work(armpl_int_t matrix_layout, char uplo,
                               armpl_int_t n, armpl_int_t kd,
                               armpl_int_t nrhs, double *ab, armpl_int_t ldab,
                               double *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbsv_work*, *LAPACKE_dpbsv_work*, *LAPACKE_spbsv_work* and *LAPACKE_zpbsv_work*.

4.19.919 LAPACKE_dpbsvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpbsvx(armpl_int_t matrix_layout, char fact, char uplo,
                           armpl_int_t n, armpl_int_t kd, armpl_int_t nrhs,
                           double *ab, armpl_int_t ldab, double *afb,
                           armpl_int_t ldafb, char *equed, double *s,
                           double *b, armpl_int_t ldb, double *x,
```

(continues on next page)

(continued from previous page)

```
                                armpl_int_t ldx, double *rcond, double *ferr,
                                double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbsvx*, *LAPACKE_dpbsvx*, *LAPACKE_spbsvx* and *LAPACKE_zpbsvx*. It also exists with a native Fortran interface as *dpbsvx*.

4.19.920 LAPACKE_dpbsvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpbsvx_work(armpl_int_t matrix_layout, char fact,
                                char uplo, armpl_int_t n, armpl_int_t kd,
                                armpl_int_t nrhs, double *ab,
                                armpl_int_t ldab, double *afb,
                                armpl_int_t ldafb, char *equed, double *s,
                                double *b, armpl_int_t ldb, double *x,
                                armpl_int_t ldx, double *rcond, double *ferr,
                                double *berr, double *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbsvx_work*, *LAPACKE_dpbsvx_work*, *LAPACKE_spbsvx_work* and *LAPACKE_zpbsvx_work*.

4.19.921 LAPACKE_dpbtrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpbtrf(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t kd, double *ab,
                           armpl_int_t ldab);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbtrf*, *LAPACKE_dpbtrf*, *LAPACKE_spbtrf* and *LAPACKE_zpbtrf*. It also exists with a native Fortran interface as *dpbtrf*.

4.19.922 LAPACKE_dpbtrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpbtrf_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t kd, double *ab,
                                armpl_int_t ldab);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbtrf_work*, *LAPACKE_dpbtrf_work*, *LAPACKE_spbtrf_work* and *LAPACKE_zpbtrf_work*.

4.19.923 LAPACKE_dpbtrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpbtrs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t kd, armpl_int_t nrhs,
                           const double *ab, armpl_int_t ldab, double *b,
                           armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbtrs*, *LAPACKE_dpbtrs*, *LAPACKE_spbtrs* and *LAPACKE_zpbtrs*. It also exists with a native Fortran interface as *dpbtrs*.

4.19.924 LAPACKE_dpbtrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpbtrs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t kd,
                                armpl_int_t nrhs, const double *ab,
                                armpl_int_t ldab, double *b,
                                armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbtrs_work*, *LAPACKE_dpbtrs_work*, *LAPACKE_spbtrs_work* and *LAPACKE_zpbtrs_work*.

4.19.925 LAPACKE_dpftrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpftrf(armpl_int_t matrix_layout, char transr, char uplo,
                           armpl_int_t n, double *a);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpftrf*, *LAPACKE_dpftrf*, *LAPACKE_spftrf* and *LAPACKE_zpftrf*. It also exists with a native Fortran interface as *dpftrf*.

4.19.926 LAPACKE_dpftrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpftrf_work(armpl_int_t matrix_layout, char transr,
                                char uplo, armpl_int_t n, double *a);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpftrf_work*, *LAPACKE_dpftrf_work*, *LAPACKE_spftrf_work* and *LAPACKE_zpftrf_work*.

4.19.927 LAPACKE_dpftri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpftri(armpl_int_t matrix_layout, char transr, char uplo,
                           armpl_int_t n, double *a);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpftri*, *LAPACKE_dpftri*, *LAPACKE_spftri* and *LAPACKE_zpftri*. It also exists with a native Fortran interface as *dpftri*.

4.19.928 LAPACKE_dpftri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpftri_work(armpl_int_t matrix_layout, char transr,
                                char uplo, armpl_int_t n, double *a);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpftri_work*, *LAPACKE_dpftri_work*, *LAPACKE_spftri_work* and *LAPACKE_zpftri_work*.

4.19.929 LAPACKE_dpftrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpftrs(armpl_int_t matrix_layout, char transr, char uplo,
                           armpl_int_t n, armpl_int_t nrhs, const double *a,
                           double *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpftrs*, *LAPACKE_dpftrs*, *LAPACKE_spftrs* and *LAPACKE_zpftrs*. It also exists with a native Fortran interface as *dpftrs*.

4.19.930 LAPACKE_dpftrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpftrs_work(armpl_int_t matrix_layout, char transr,
                                char uplo, armpl_int_t n, armpl_int_t nrhs,
                                const double *a, double *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpftrs_work*, *LAPACKE_dpftrs_work*, *LAPACKE_spftrs_work* and *LAPACKE_zpftrs_work*.

4.19.931 LAPACKE_dpocon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpocon(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, const double *a, armpl_int_t lda,
                           double anorm, double *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpocon*, *LAPACKE_dpocon*, *LAPACKE_spocon* and *LAPACKE_zpocon*. It also exists with a native Fortran interface as *dpocon*.

4.19.932 LAPACKE_dpocon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpocon_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, const double *a,
                                armpl_int_t lda, double anorm, double *rcond,
                                double *work, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpocon_work*, *LAPACKE_dpocon_work*, *LAPACKE_spocon_work* and *LAPACKE_zpocon_work*.

4.19.933 LAPACKE_dpoequ

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpoequ(armpl_int_t matrix_layout, armpl_int_t n,
                           const double *a, armpl_int_t lda, double *s,
                           double *scond, double *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpoequ*, *LAPACKE_dpoequ*, *LAPACKE_spoequ* and *LAPACKE_zpoequ*. It also exists with a native Fortran interface as *dpoequ*.

4.19.934 LAPACKE_dpoequ_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpoequ_work(armpl_int_t matrix_layout, armpl_int_t n,
                                const double *a, armpl_int_t lda, double *s,
                                double *scond, double *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpoequ_work*, *LAPACKE_dpoequ_work*, *LAPACKE_spoequ_work* and *LAPACKE_zpoequ_work*.

4.19.935 LAPACKE_dpoequb

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpoequb(armpl_int_t matrix_layout, armpl_int_t n,
                            const double *a, armpl_int_t lda, double *s,
                            double *scond, double *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpoequb*, *LAPACKE_dpoequb*, *LAPACKE_spoequb* and *LAPACKE_zpoequb*. It also exists with a native Fortran interface as *dpoequb*.

4.19.936 LAPACKE_dpoequb_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpoequb_work(armpl_int_t matrix_layout, armpl_int_t n,
                                 const double *a, armpl_int_t lda, double *s,
                                 double *scond, double *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpoequb_work*, *LAPACKE_dpoequb_work*, *LAPACKE_spoequb_work* and *LAPACKE_zpoequb_work*.

4.19.937 LAPACKE_dporfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dporfs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs, const double *a,
                           armpl_int_t lda, const double *af,
                           armpl_int_t ldaf, const double *b, armpl_int_t ldb,
                           double *x, armpl_int_t ldx, double *ferr,
                           double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cporfs*, *LAPACKE_dporfs*, *LAPACKE_sporfs* and *LAPACKE_zporfs*. It also exists with a native Fortran interface as *dporfs*.

4.19.938 LAPACKE_dporfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dporfs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const double *a, armpl_int_t lda,
                                const double *af, armpl_int_t ldaf,
                                const double *b, armpl_int_t ldb, double *x,
                                armpl_int_t ldx, double *ferr, double *berr,
                                double *work, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cporfs_work*, *LAPACKE_dporfs_work*, *LAPACKE_sporfs_work* and *LAPACKE_zporfs_work*.

4.19.939 LAPACKE_dporfsx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dporfsx(armpl_int_t matrix_layout, char uplo, char equed,
                            armpl_int_t n, armpl_int_t nrhs, const double *a,
                            armpl_int_t lda, const double *af,
                            armpl_int_t ldaf, const double *s,
                            const double *b, armpl_int_t ldb, double *x,
                            armpl_int_t ldx, double *rcond, double *berr,
                            armpl_int_t n_err_bnds, double *err_bnds_norm,
                            double *err_bnds_comp, armpl_int_t nparams,
                            double *params);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cporfsx*, *LAPACKE_dporfsx*, *LAPACKE_sporfsx* and *LAPACKE_zporfsx*. It also exists with a native Fortran interface as *dporfsx*.

4.19.940 LAPACKE_dporfsx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dporfsx_work(armpl_int_t matrix_layout, char uplo,
                                 char equed, armpl_int_t n, armpl_int_t nrhs,
                                 const double *a, armpl_int_t lda,
                                 const double *af, armpl_int_t ldaf,
                                 const double *s, const double *b,
                                 armpl_int_t ldb, double *x, armpl_int_t ldx,
                                 double *rcond, double *berr,
                                 armpl_int_t n_err_bnds,
                                 double *err_bnds_norm, double *err_bnds_comp,
                                 armpl_int_t nparams, double *params,
                                 double *work, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cporfsx_work*, *LAPACKE_dporfsx_work*, *LAPACKE_sporfsx_work* and *LAPACKE_zporfsx_work*.

4.19.941 LAPACKE_dposv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dposv(armpl_int_t matrix_layout, char uplo, armpl_int_t n,
                          armpl_int_t nrhs, double *a, armpl_int_t lda,
                          double *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cposv*, *LAPACKE_dposv*, *LAPACKE_sposv* and *LAPACKE_zposv*. It also exists with a native Fortran interface as *dposv*.

4.19.942 LAPACKE_dposv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dposv_work(armpl_int_t matrix_layout, char uplo,
                               armpl_int_t n, armpl_int_t nrhs, double *a,
                               armpl_int_t lda, double *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cposv_work*, *LAPACKE_dposv_work*, *LAPACKE_sposv_work* and *LAPACKE_zposv_work*.

4.19.943 LAPACKE_dposvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dposvx(armpl_int_t matrix_layout, char fact, char uplo,
                           armpl_int_t n, armpl_int_t nrhs, double *a,
                           armpl_int_t lda, double *af, armpl_int_t ldaf,
                           char *equed, double *s, double *b, armpl_int_t ldb,
                           double *x, armpl_int_t ldx, double *rcond,
                           double *ferr, double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cposvx*, *LAPACKE_dposvx*, *LAPACKE_sposvx* and *LAPACKE_zposvx*. It also exists with a native Fortran interface as *dposvx*.

4.19.944 LAPACKE_dposvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dposvx_work(armpl_int_t matrix_layout, char fact,
                                char uplo, armpl_int_t n, armpl_int_t nrhs,
                                double *a, armpl_int_t lda, double *af,
```

(continues on next page)

(continued from previous page)

```
                                armpl_int_t ldaf, char *equed, double *s,
                                double *b, armpl_int_t ldb, double *x,
                                armpl_int_t ldx, double *rcond, double *ferr,
                                double *berr, double *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cposvx_work*, *LAPACKE_dposvx_work*, *LAPACKE_sposvx_work* and *LAPACKE_zposvx_work*.

4.19.945 LAPACKE_dposvxx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dposvxx(armpl_int_t matrix_layout, char fact, char uplo,
                            armpl_int_t n, armpl_int_t nrhs, double *a,
                            armpl_int_t lda, double *af, armpl_int_t ldaf,
                            char *equed, double *s, double *b,
                            armpl_int_t ldb, double *x, armpl_int_t ldx,
                            double *rcond, double *rpvgrw, double *berr,
                            armpl_int_t n_err_bnds, double *err_bnds_norm,
                            double *err_bnds_comp, armpl_int_t nparams,
                            double *params);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cposvxx*, *LAPACKE_dposvxx*, *LAPACKE_sposvxx* and *LAPACKE_zposvxx*. It also exists with a native Fortran interface as *dposvxx*.

4.19.946 LAPACKE_dposvxx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dposvxx_work(armpl_int_t matrix_layout, char fact,
                                 char uplo, armpl_int_t n, armpl_int_t nrhs,
                                 double *a, armpl_int_t lda, double *af,
                                 armpl_int_t ldaf, char *equed, double *s,
                                 double *b, armpl_int_t ldb, double *x,
                                 armpl_int_t ldx, double *rcond,
                                 double *rpvgrw, double *berr,
                                 armpl_int_t n_err_bnds,
                                 double *err_bnds_norm, double *err_bnds_comp,
                                 armpl_int_t nparams, double *params,
                                 double *work, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cposvxx_work*, *LAPACKE_dposvxx_work*, *LAPACKE_sposvxx_work* and *LAPACKE_zposvxx_work*.

4.19.947 LAPACKE_dpotrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpotrf(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, double *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpotrf*, *LAPACKE_dpotrf*, *LAPACKE_spotrf* and *LAPACKE_zpotrf*. It also exists with a native Fortran interface as *dpotrf*.

4.19.948 LAPACKE_dpotrf2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpotrf2(armpl_int_t matrix_layout, char uplo,
                            armpl_int_t n, double *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpotrf2*, *LAPACKE_dpotrf2*, *LAPACKE_spotrf2* and *LAPACKE_zpotrf2*. It also exists with a native Fortran interface as *dpotrf2*.

4.19.949 LAPACKE_dpotrf2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpotrf2_work(armpl_int_t matrix_layout, char uplo,
                                 armpl_int_t n, double *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpotrf2_work*, *LAPACKE_dpotrf2_work*, *LAPACKE_spotrf2_work* and *LAPACKE_zpotrf2_work*.

4.19.950 LAPACKE_dpotrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpotrf_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, double *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpotrf_work*, *LAPACKE_dpotrf_work*, *LAPACKE_spotrf_work* and *LAPACKE_zpotrf_work*.

4.19.951 LAPACKE_dpotri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpotri(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, double *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpotri*, *LAPACKE_dpotri*, *LAPACKE_spotri* and *LAPACKE_zpotri*. It also exists with a native Fortran interface as *dpotri*.

4.19.952 LAPACKE_dpotri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpotri_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, double *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpotri_work*, *LAPACKE_dpotri_work*, *LAPACKE_spotri_work* and *LAPACKE_zpotri_work*.

4.19.953 LAPACKE_dpotrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpotrs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs, const double *a,
                           armpl_int_t lda, double *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpotrs*, *LAPACKE_dpotrs*, *LAPACKE_spotrs* and *LAPACKE_zpotrs*. It also exists with a native Fortran interface as *dpotrs*.

4.19.954 LAPACKE_dpotrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpotrs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const double *a, armpl_int_t lda, double *b,
                                armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpotrs_work*, *LAPACKE_dpotrs_work*, *LAPACKE_spotrs_work* and *LAPACKE_zpotrs_work*.

4.19.955 LAPACKE_dppcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dppcon(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, const double *ap, double anorm,
                           double *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cppcon*, *LAPACKE_dppcon*, *LAPACKE_sppcon* and *LAPACKE_zppcon*. It also exists with a native Fortran interface as *dppcon*.

4.19.956 LAPACKE_dppcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dppcon_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, const double *ap, double anorm,
                                double *rcond, double *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cppcon_work*, *LAPACKE_dppcon_work*, *LAPACKE_sppcon_work* and *LAPACKE_zppcon_work*.

4.19.957 LAPACKE_dppequ

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dppequ(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, const double *ap, double *s,
                           double *scond, double *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cppequ*, *LAPACKE_dppequ*, *LAPACKE_sppequ* and *LAPACKE_zppequ*. It also exists with a native Fortran interface as *dppequ*.

4.19.958 LAPACKE_dppequ_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dppequ_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, const double *ap, double *s,
                                double *scond, double *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cppequ_work*, *LAPACKE_dppequ_work*, *LAPACKE_sppequ_work* and *LAPACKE_zppequ_work*.

4.19.959 LAPACKE_dpprfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpprfs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs, const double *ap,
                           const double *afp, const double *b,
                           armpl_int_t ldb, double *x, armpl_int_t ldx,
                           double *ferr, double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpprfs*, *LAPACKE_dpprfs*, *LAPACKE_spprfs* and *LAPACKE_zpprfs*. It also exists with a native Fortran interface as *dpprfs*.

4.19.960 LAPACKE_dpprfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpprfs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const double *ap, const double *afp,
                                const double *b, armpl_int_t ldb, double *x,
                                armpl_int_t ldx, double *ferr, double *berr,
                                double *work, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpprfs_work*, *LAPACKE_dpprfs_work*, *LAPACKE_spprfs_work* and *LAPACKE_zpprfs_work*.

4.19.961 LAPACKE_dppsv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dppsv(armpl_int_t matrix_layout, char uplo, armpl_int_t n,
                          armpl_int_t nrhs, double *ap, double *b,
                          armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cppsv*, *LAPACKE_dppsv*, *LAPACKE_sppsv* and *LAPACKE_zppsv*. It also exists with a native Fortran interface as *dppsv*.

4.19.962 LAPACKE_dppsv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dppsv_work(armpl_int_t matrix_layout, char uplo,
                               armpl_int_t n, armpl_int_t nrhs, double *ap,
                               double *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cppsv_work*, *LAPACKE_dppsv_work*, *LAPACKE_sppsv_work* and *LAPACKE_zppsv_work*.

4.19.963 LAPACKE_dppsvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dppsvx(armpl_int_t matrix_layout, char fact, char uplo,
                           armpl_int_t n, armpl_int_t nrhs, double *ap,
                           double *afp, char *equed, double *s, double *b,
                           armpl_int_t ldb, double *x, armpl_int_t ldx,
                           double *rcond, double *ferr, double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cppsvx*, *LAPACKE_dppsvx*, *LAPACKE_sppsvx* and *LAPACKE_zppsvx*. It also exists with a native Fortran interface as *dppsvx*.

4.19.964 LAPACKE_dppsvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dppsvx_work(armpl_int_t matrix_layout, char fact,
                                char uplo, armpl_int_t n, armpl_int_t nrhs,
                                double *ap, double *afp, char *equed,
                                double *s, double *b, armpl_int_t ldb,
                                double *x, armpl_int_t ldx, double *rcond,
                                double *ferr, double *berr, double *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cppsvx_work*, *LAPACKE_dppsvx_work*, *LAPACKE_sppsvx_work* and *LAPACKE_zppsvx_work*.

4.19.965 LAPACKE_dpptrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpptrf(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, double *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpptrf*, *LAPACKE_dpptrf*, *LAPACKE_spptrf* and *LAPACKE_zpptrf*. It also exists with a native Fortran interface as *dpptrf*.

4.19.966 LAPACKE_dpptrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpptrf_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, double *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpptrf_work*, *LAPACKE_dpptrf_work*, *LAPACKE_spptrf_work* and *LAPACKE_zpptrf_work*.

4.19.967 LAPACKE_dpptri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpptri(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, double *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpptri*, *LAPACKE_dpptri*, *LAPACKE_spptri* and *LAPACKE_zpptri*. It also exists with a native Fortran interface as *dpptri*.

4.19.968 LAPACKE_dpptri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpptri_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, double *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpptri_work*, *LAPACKE_dpptri_work*, *LAPACKE_spptri_work* and *LAPACKE_zpptri_work*.

4.19.969 LAPACKE_dpptrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpptrs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs, const double *ap,
                           double *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpptrs*, *LAPACKE_dpptrs*, *LAPACKE_spptrs* and *LAPACKE_zpptrs*. It also exists with a native Fortran interface as *dpptrs*.

4.19.970 LAPACKE_dpptrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpptrs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const double *ap, double *b,
                                armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpptrs_work*, *LAPACKE_dpptrs_work*, *LAPACKE_spptrs_work* and *LAPACKE_zpptrs_work*.

4.19.971 LAPACKE_dpstrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpstrf(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, double *a, armpl_int_t lda,
                           armpl_int_t *piv, armpl_int_t *rank, double tol);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpstrf*, *LAPACKE_dpstrf*, *LAPACKE_spstrf* and *LAPACKE_zpstrf*. It also exists with a native Fortran interface as *dpstrf*.

4.19.972 LAPACKE_dpstrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpstrf_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, double *a, armpl_int_t lda,
                                armpl_int_t *piv, armpl_int_t *rank,
                                double tol, double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpstrf_work*, *LAPACKE_dpstrf_work*, *LAPACKE_spstrf_work* and *LAPACKE_zpstrf_work*.

4.19.973 LAPACKE_dptcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dptcon(armpl_int_t n, const double *d, const double *e,
                           double anorm, double *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.974 LAPACKE_dptcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dptcon_work(armpl_int_t n, const double *d,
                                const double *e, double anorm, double *rcond,
                                double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cptcon_work*, *LAPACKE_dptcon_work*, *LAPACKE_sptcon_work* and *LAPACKE_zptcon_work*.

4.19.975 LAPACKE_dpteqr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpteqr(armpl_int_t matrix_layout, char compz,
                           armpl_int_t n, double *d, double *e, double *z,
                           armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpteqr*, *LAPACKE_dpteqr*, *LAPACKE_spteqr* and *LAPACKE_zpteqr*. It also exists with a native Fortran interface as *dpteqr*.

4.19.976 LAPACKE_dpteqr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpteqr_work(armpl_int_t matrix_layout, char compz,
                                armpl_int_t n, double *d, double *e,
                                double *z, armpl_int_t ldz, double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpteqr_work*, *LAPACKE_dpteqr_work*, *LAPACKE_spteqr_work* and *LAPACKE_zpteqr_work*.

4.19.977 LAPACKE_dptrfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dptrfs(armpl_int_t matrix_layout, armpl_int_t n,
                           armpl_int_t nrhs, const double *d, const double *e,
                           const double *df, const double *ef,
                           const double *b, armpl_int_t ldb, double *x,
                           armpl_int_t ldx, double *ferr, double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cptrfs*, *LAPACKE_dptrfs*, *LAPACKE_sptrfs* and *LAPACKE_zptrfs*. It also exists with a native Fortran interface as *dptrfs*.

4.19.978 LAPACKE_dptrfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dptrfs_work(armpl_int_t matrix_layout, armpl_int_t n,
                                armpl_int_t nrhs, const double *d,
                                const double *e, const double *df,
                                const double *ef, const double *b,
                                armpl_int_t ldb, double *x, armpl_int_t ldx,
                                double *ferr, double *berr, double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cptrfs_work*, *LAPACKE_dptrfs_work*, *LAPACKE_sptrfs_work* and *LAPACKE_zptrfs_work*.

4.19.979 LAPACKE_dptsv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dptsv(armpl_int_t matrix_layout, armpl_int_t n,
                          armpl_int_t nrhs, double *d, double *e, double *b,
                          armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cptsv*, *LAPACKE_dptsv*, *LAPACKE_sptsv* and *LAPACKE_zptsv*. It also exists with a native Fortran interface as *dptsv*.

4.19.980 LAPACKE_dptsv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dptsv_work(armpl_int_t matrix_layout, armpl_int_t n,
                               armpl_int_t nrhs, double *d, double *e,
                               double *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cptsv_work*, *LAPACKE_dptsv_work*, *LAPACKE_sptsv_work* and *LAPACKE_zptsv_work*.

4.19.981 LAPACKE_dptsvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dptsvx(armpl_int_t matrix_layout, char fact,
                           armpl_int_t n, armpl_int_t nrhs, const double *d,
                           const double *e, double *df, double *ef,
                           const double *b, armpl_int_t ldb, double *x,
                           armpl_int_t ldx, double *rcond, double *ferr,
                           double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cptsvx*, *LAPACKE_dptsvx*, *LAPACKE_sptsvx* and *LAPACKE_zptsvx*. It also exists with a native Fortran interface as *dptsvx*.

4.19.982 LAPACKE_dptsvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dptsvx_work(armpl_int_t matrix_layout, char fact,
                                armpl_int_t n, armpl_int_t nrhs,
                                const double *d, const double *e, double *df,
                                double *ef, const double *b, armpl_int_t ldb,
                                double *x, armpl_int_t ldx, double *rcond,
                                double *ferr, double *berr, double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cptsvx_work*, *LAPACKE_dptsvx_work*, *LAPACKE_sptsvx_work* and *LAPACKE_zptsvx_work*.

4.19.983 LAPACKE_dpttrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpttrf(armpl_int_t n, double *d, double *e);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpttrf*, *LAPACKE_dpttrf*, *LAPACKE_spttrf* and *LAPACKE_zpttrf*. It also exists with a native Fortran interface as *dpttrf*.

4.19.984 LAPACKE_dpttrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpttrf_work(armpl_int_t n, double *d, double *e);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpttrf_work*, *LAPACKE_dpttrf_work*, *LAPACKE_spttrf_work* and *LAPACKE_zpttrf_work*.

4.19.985 LAPACKE_dpttrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpttrs(armpl_int_t matrix_layout, armpl_int_t n,
                           armpl_int_t nrhs, const double *d, const double *e,
                           double *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpttrs*, *LAPACKE_dpttrs*, *LAPACKE_spttrs* and *LAPACKE_zpttrs*. It also exists with a native Fortran interface as *dpttrs*.

4.19.986 LAPACKE_dpttrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dpttrs_work(armpl_int_t matrix_layout, armpl_int_t n,
                                armpl_int_t nrhs, const double *d,
                                const double *e, double *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpttrs_work*, *LAPACKE_dpttrs_work*, *LAPACKE_spttrs_work* and *LAPACKE_zpttrs_work*.

4.19.987 LAPACKE_dsbev

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsbev(armpl_int_t matrix_layout, char jobz, char uplo,
                          armpl_int_t n, armpl_int_t kd, double *ab,
                          armpl_int_t ldab, double *w, double *z,
                          armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbev* and *LAPACKE_ssbev*. It also exists with a native Fortran interface as *dsbev*.

4.19.988 LAPACKE_dsbev_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsbev_2stage(armpl_int_t matrix_layout, char jobz,
                                 char uplo, armpl_int_t n, armpl_int_t kd,
                                 double *ab, armpl_int_t ldab, double *w,
                                 double *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbev_2stage* and *LAPACKE_ssbev_2stage*. It also exists with a native Fortran interface as *dsbev_2stage*.

4.19.989 LAPACKE_dsbev_2stage_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsbev_2stage_work(armpl_int_t matrix_layout, char jobz,
                                      char uplo, armpl_int_t n,
                                      armpl_int_t kd, double *ab,
                                      armpl_int_t ldab, double *w, double *z,
                                      armpl_int_t ldz, double *work,
                                      armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbev_2stage_work* and *LAPACKE_ssbev_2stage_work*.

4.19.990 LAPACKE_dsbev_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsbev_work(armpl_int_t matrix_layout, char jobz,
                               char uplo, armpl_int_t n, armpl_int_t kd,
                               double *ab, armpl_int_t ldab, double *w,
                               double *z, armpl_int_t ldz, double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbev_work* and *LAPACKE_ssbev_work*.

4.19.991 LAPACKE_dsbevd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsbevd(armpl_int_t matrix_layout, char jobz, char uplo,
                           armpl_int_t n, armpl_int_t kd, double *ab,
                           armpl_int_t ldab, double *w, double *z,
                           armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbevd* and *LAPACKE_ssbevd*. It also exists with a native Fortran interface as *dsbevd*.

4.19.992 LAPACKE_dsbevd_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsbevd_2stage(armpl_int_t matrix_layout, char jobz,
                                  char uplo, armpl_int_t n, armpl_int_t kd,
                                  double *ab, armpl_int_t ldab, double *w,
                                  double *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbevd_2stage* and *LAPACKE_ssbevd_2stage*. It also exists with a native Fortran interface as *dsbevd_2stage*.

4.19.993 LAPACKE_dsbevd_2stage_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsbevd_2stage_work(armpl_int_t matrix_layout, char jobz,
                                       char uplo, armpl_int_t n,
                                       armpl_int_t kd, double *ab,
                                       armpl_int_t ldab, double *w, double *z,
                                       armpl_int_t ldz, double *work,
                                       armpl_int_t lwork, armpl_int_t *iwork,
                                       armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbevd_2stage_work* and *LAPACKE_ssbevd_2stage_work*.

4.19.994 LAPACKE_dsbevd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsbevd_work(armpl_int_t matrix_layout, char jobz,
                                char uplo, armpl_int_t n, armpl_int_t kd,
                                double *ab, armpl_int_t ldab, double *w,
                                double *z, armpl_int_t ldz, double *work,
                                armpl_int_t lwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbevd_work* and *LAPACKE_ssbevd_work*.

4.19.995 LAPACKE_dsbevx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsbevx(armpl_int_t matrix_layout, char jobz, char range,
                           char uplo, armpl_int_t n, armpl_int_t kd,
                           double *ab, armpl_int_t ldab, double *q,
                           armpl_int_t ldq, double vl, double vu,
                           armpl_int_t il, armpl_int_t iu, double abstol,
                           armpl_int_t *m, double *w, double *z,
                           armpl_int_t ldz, armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbevx* and *LAPACKE_ssbevx*. It also exists with a native Fortran interface as *dsbevx*.

4.19.996 LAPACKE_dsbevx_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsbevx_2stage(armpl_int_t matrix_layout, char jobz,
                                  char range, char uplo, armpl_int_t n,
                                  armpl_int_t kd, double *ab,
                                  armpl_int_t ldab, double *q,
                                  armpl_int_t ldq, double vl, double vu,
                                  armpl_int_t il, armpl_int_t iu,
                                  double abstol, armpl_int_t *m, double *w,
                                  double *z, armpl_int_t ldz,
                                  armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbevx_2stage* and *LAPACKE_ssbevx_2stage*. It also exists with a native Fortran interface as *dsbevx_2stage*.

4.19.997 LAPACKE_dsbevx_2stage_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsbevx_2stage_work(armpl_int_t matrix_layout, char jobz,
                                       char range, char uplo, armpl_int_t n,
                                       armpl_int_t kd, double *ab,
                                       armpl_int_t ldab, double *q,
                                       armpl_int_t ldq, double vl, double vu,
                                       armpl_int_t il, armpl_int_t iu,
                                       double abstol, armpl_int_t *m,
```

(continues on next page)

(continued from previous page)

```
                                        double *w, double *z, armpl_int_t ldz,
                                        double *work, armpl_int_t lwork,
                                        armpl_int_t *iwork,
                                        armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbevx_2stage_work* and *LAPACKE_ssbevx_2stage_work*.

4.19.998 LAPACKE_dsbevx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsbevx_work(armpl_int_t matrix_layout, char jobz,
                                char range, char uplo, armpl_int_t n,
                                armpl_int_t kd, double *ab, armpl_int_t ldab,
                                double *q, armpl_int_t ldq, double vl,
                                double vu, armpl_int_t il, armpl_int_t iu,
                                double abstol, armpl_int_t *m, double *w,
                                double *z, armpl_int_t ldz, double *work,
                                armpl_int_t *iwork, armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbevx_work* and *LAPACKE_ssbevx_work*.

4.19.999 LAPACKE_dsbgst

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsbgst(armpl_int_t matrix_layout, char vect, char uplo,
                           armpl_int_t n, armpl_int_t ka, armpl_int_t kb,
                           double *ab, armpl_int_t ldab, const double *bb,
                           armpl_int_t ldbb, double *x, armpl_int_t ldx);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbgst* and *LAPACKE_ssbgst*. It also exists with a native Fortran interface as *dsbgst*.

4.19.1000 LAPACKE_dsbgst_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsbgst_work(armpl_int_t matrix_layout, char vect,
                                char uplo, armpl_int_t n, armpl_int_t ka,
                                armpl_int_t kb, double *ab, armpl_int_t ldab,
                                const double *bb, armpl_int_t ldbb, double *x,
                                armpl_int_t ldx, double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbgst_work* and *LAPACKE_ssbgst_work*.

4.19.1001 LAPACKE_dsbgv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsbgv(armpl_int_t matrix_layout, char jobz, char uplo,
                          armpl_int_t n, armpl_int_t ka, armpl_int_t kb,
                          double *ab, armpl_int_t ldab, double *bb,
                          armpl_int_t ldbb, double *w, double *z,
                          armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbgv* and *LAPACKE_ssbgv*. It also exists with a native Fortran interface as *dsbgv*.

4.19.1002 LAPACKE_dsbgv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsbgv_work(armpl_int_t matrix_layout, char jobz,
                               char uplo, armpl_int_t n, armpl_int_t ka,
                               armpl_int_t kb, double *ab, armpl_int_t ldab,
                               double *bb, armpl_int_t ldbb, double *w,
                               double *z, armpl_int_t ldz, double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbgv_work* and *LAPACKE_ssbgv_work*.

4.19.1003 LAPACKE_dsbgvd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsbgvd(armpl_int_t matrix_layout, char jobz, char uplo,
                           armpl_int_t n, armpl_int_t ka, armpl_int_t kb,
                           double *ab, armpl_int_t ldab, double *bb,
                           armpl_int_t ldbb, double *w, double *z,
                           armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbgvd* and *LAPACKE_ssbgvd*. It also exists with a native Fortran interface as *dsbgvd*.

4.19.1004 LAPACKE_dsbgvd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsbgvd_work(armpl_int_t matrix_layout, char jobz,
                                char uplo, armpl_int_t n, armpl_int_t ka,
                                armpl_int_t kb, double *ab, armpl_int_t ldab,
                                double *bb, armpl_int_t ldbb, double *w,
                                double *z, armpl_int_t ldz, double *work,
                                armpl_int_t lwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.1005 LAPACKE_dsbgvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsbgvx(armpl_int_t matrix_layout, char jobz, char range,
                           char uplo, armpl_int_t n, armpl_int_t ka,
                           armpl_int_t kb, double *ab, armpl_int_t ldab,
                           double *bb, armpl_int_t ldbb, double *q,
                           armpl_int_t ldq, double vl, double vu,
                           armpl_int_t il, armpl_int_t iu, double abstol,
                           armpl_int_t *m, double *w, double *z,
                           armpl_int_t ldz, armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbgvx* and *LAPACKE_ssbgvx*. It also exists with a native Fortran interface as *dsbgvx*.

4.19.1006 LAPACKE_dsbgvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsbgvx_work(armpl_int_t matrix_layout, char jobz,
                                char range, char uplo, armpl_int_t n,
                                armpl_int_t ka, armpl_int_t kb, double *ab,
                                armpl_int_t ldab, double *bb,
                                armpl_int_t ldbb, double *q, armpl_int_t ldq,
                                double vl, double vu, armpl_int_t il,
                                armpl_int_t iu, double abstol, armpl_int_t *m,
                                double *w, double *z, armpl_int_t ldz,
                                double *work, armpl_int_t *iwork,
                                armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbgvx_work* and *LAPACKE_ssbgvx_work*.

4.19.1007 LAPACKE_dsbtrd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsbtrd(armpl_int_t matrix_layout, char vect, char uplo,
                           armpl_int_t n, armpl_int_t kd, double *ab,
                           armpl_int_t ldab, double *d, double *e, double *q,
                           armpl_int_t ldq);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbtrd* and *LAPACKE_ssbtrd*. It also exists with a native Fortran interface as *dsbtrd*.

4.19.1008 LAPACKE_dsbtrd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsbtrd_work(armpl_int_t matrix_layout, char vect,
                                char uplo, armpl_int_t n, armpl_int_t kd,
                                double *ab, armpl_int_t ldab, double *d,
                                double *e, double *q, armpl_int_t ldq,
                                double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbtrd_work* and *LAPACKE_ssbtrd_work*.

4.19.1009 LAPACKE_dsfrk

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsfrk(armpl_int_t matrix_layout, char transr, char uplo,
                          char trans, armpl_int_t n, armpl_int_t k,
                          double alpha, const double *a, armpl_int_t lda,
                          double beta, double *c);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsfrk* and *LAPACKE_ssfrk*. It also exists with a native Fortran interface as *dsfrk*.

4.19.1010 LAPACKE_dsfrk_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsfrk_work(armpl_int_t matrix_layout, char transr,
                               char uplo, char trans, armpl_int_t n,
                               armpl_int_t k, double alpha, const double *a,
                               armpl_int_t lda, double beta, double *c);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsfrk_work* and *LAPACKE_ssfrk_work*.

4.19.1011 LAPACKE_dsgesv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsgesv(armpl_int_t matrix_layout, armpl_int_t n,
                           armpl_int_t nrhs, double *a, armpl_int_t lda,
                           armpl_int_t *ipiv, double *b, armpl_int_t ldb,
                           double *x, armpl_int_t ldx, armpl_int_t *iter);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsgesv*. It also exists with a native Fortran interface as *dsgesv*.

4.19.1012 LAPACKE_dsgesv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsgesv_work(armpl_int_t matrix_layout, armpl_int_t n,
                                armpl_int_t nrhs, double *a, armpl_int_t lda,
                                armpl_int_t *ipiv, double *b, armpl_int_t ldb,
                                double *x, armpl_int_t ldx, double *work,
                                float *swork, armpl_int_t *iter);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsgesv_work*.

4.19.1013 LAPACKE_dspcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dspcon(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, const double *ap,
                           const armpl_int_t *ipiv, double anorm,
                           double *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cspcon*, *LAPACKE_dspcon*, *LAPACKE_sspcon* and *LAPACKE_zspcon*. It also exists with a native Fortran interface as *dspcon*.

4.19.1014 LAPACKE_dspcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dspcon_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, const double *ap,
                                const armpl_int_t *ipiv, double anorm,
                                double *rcond, double *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cspcon_work*, *LAPACKE_dspcon_work*, *LAPACKE_sspcon_work* and *LAPACKE_zspcon_work*.

4.19.1015 LAPACKE_dspev

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dspev(armpl_int_t matrix_layout, char jobz, char uplo,
                          armpl_int_t n, double *ap, double *w, double *z,
                          armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dspev* and *LAPACKE_sspev*. It also exists with a native Fortran interface as *dspev*.

4.19.1016 LAPACKE_dspev_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dspev_work(armpl_int_t matrix_layout, char jobz,
                               char uplo, armpl_int_t n, double *ap,
                               double *w, double *z, armpl_int_t ldz,
                               double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dspev_work* and *LAPACKE_sspev_work*.

4.19.1017 LAPACKE_dspevd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dspevd(armpl_int_t matrix_layout, char jobz, char uplo,
                           armpl_int_t n, double *ap, double *w, double *z,
                           armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dspevd* and *LAPACKE_sspevd*. It also exists with a native Fortran interface as *dspevd*.

4.19.1018 LAPACKE_dspevd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dspevd_work(armpl_int_t matrix_layout, char jobz,
                                char uplo, armpl_int_t n, double *ap,
                                double *w, double *z, armpl_int_t ldz,
                                double *work, armpl_int_t lwork,
                                armpl_int_t *iwork, armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dspevd_work* and *LAPACKE_sspevd_work*.

4.19.1019 LAPACKE_dspevx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dspevx(armpl_int_t matrix_layout, char jobz, char range,
                           char uplo, armpl_int_t n, double *ap, double vl,
                           double vu, armpl_int_t il, armpl_int_t iu,
                           double abstol, armpl_int_t *m, double *w,
                           double *z, armpl_int_t ldz, armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dspevx* and *LAPACKE_sspevx*. It also exists with a native Fortran interface as *dspevx*.

4.19.1020 LAPACKE_dspevx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dspevx_work(armpl_int_t matrix_layout, char jobz,
                                char range, char uplo, armpl_int_t n,
                                double *ap, double vl, double vu,
                                armpl_int_t il, armpl_int_t iu, double abstol,
                                armpl_int_t *m, double *w, double *z,
                                armpl_int_t ldz, double *work,
                                armpl_int_t *iwork, armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dspevx_work* and *LAPACKE_sspevx_work*.

4.19.1021 LAPACKE_dspgst

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dspgst(armpl_int_t matrix_layout, armpl_int_t itype,
                           char uplo, armpl_int_t n, double *ap,
                           const double *bp);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dspgst* and *LAPACKE_sspgst*. It also exists with a native Fortran interface as *dspgst*.

4.19.1022 LAPACKE_dspgst_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dspgst_work(armpl_int_t matrix_layout, armpl_int_t itype,
                                char uplo, armpl_int_t n, double *ap,
                                const double *bp);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dspgst_work* and *LAPACKE_sspgst_work*.

4.19.1023 LAPACKE_dspgv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dspgv(armpl_int_t matrix_layout, armpl_int_t itype,
                          char jobz, char uplo, armpl_int_t n, double *ap,
                          double *bp, double *w, double *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dspgv* and *LAPACKE_sspgv*. It also exists with a native Fortran interface as *dspgv*.

4.19.1024 LAPACKE_dspgv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dspgv_work(armpl_int_t matrix_layout, armpl_int_t itype,
                               char jobz, char uplo, armpl_int_t n,
                               double *ap, double *bp, double *w, double *z,
                               armpl_int_t ldz, double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dspgv_work* and *LAPACKE_sspgv_work*.

4.19.1025 LAPACKE_dspgvd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dspgvd(armpl_int_t matrix_layout, armpl_int_t itype,
                           char jobz, char uplo, armpl_int_t n, double *ap,
                           double *bp, double *w, double *z,
                           armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dspgvd* and *LAPACKE_sspgvd*. It also exists with a native Fortran interface as *dspgvd*.

4.19.1026 LAPACKE_dspgvd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dspgvd_work(armpl_int_t matrix_layout, armpl_int_t itype,
                                char jobz, char uplo, armpl_int_t n,
                                double *ap, double *bp, double *w, double *z,
                                armpl_int_t ldz, double *work,
                                armpl_int_t lwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dspgvd_work* and *LAPACKE_sspgvd_work*.

4.19.1027 LAPACKE_dspgvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dspgvx(armpl_int_t matrix_layout, armpl_int_t itype,
                           char jobz, char range, char uplo, armpl_int_t n,
                           double *ap, double *bp, double vl, double vu,
                           armpl_int_t il, armpl_int_t iu, double abstol,
                           armpl_int_t *m, double *w, double *z,
                           armpl_int_t ldz, armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dspgvx* and *LAPACKE_sspgvx*. It also exists with a native Fortran interface as *dspgvx*.

4.19.1028 LAPACKE_dspgvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dspgvx_work(armpl_int_t matrix_layout, armpl_int_t itype,
                                char jobz, char range, char uplo,
                                armpl_int_t n, double *ap, double *bp,
                                double vl, double vu, armpl_int_t il,
```

(continues on next page)

(continued from previous page)

```
                                armpl_int_t iu, double abstol, armpl_int_t *m,
                                double *w, double *z, armpl_int_t ldz,
                                double *work, armpl_int_t *iwork,
                                armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dspgvx_work* and *LAPACKE_sspgvx_work*.

4.19.1029 LAPACKE_dsposv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsposv(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs, double *a,
                           armpl_int_t lda, double *b, armpl_int_t ldb,
                           double *x, armpl_int_t ldx, armpl_int_t *iter);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsposv*. It also exists with a native Fortran interface as *dsposv*.

4.19.1030 LAPACKE_dsposv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsposv_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs, double *a,
                                armpl_int_t lda, double *b, armpl_int_t ldb,
                                double *x, armpl_int_t ldx, double *work,
                                float *swork, armpl_int_t *iter);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.1031 LAPACKE_dsprfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsprfs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs, const double *ap,
                           const double *afp, const armpl_int_t *ipiv,
                           const double *b, armpl_int_t ldb, double *x,
                           armpl_int_t ldx, double *ferr, double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csprfs*, *LAPACKE_dsprfs*, *LAPACKE_ssprfs* and *LAPACKE_zsprfs*. It also exists with a native Fortran interface as *dsprfs*.

4.19.1032 LAPACKE_dsprfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsprfs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const double *ap, const double *afp,
                                const armpl_int_t *ipiv, const double *b,
                                armpl_int_t ldb, double *x, armpl_int_t ldx,
                                double *ferr, double *berr, double *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csprfs_work*, *LAPACKE_dsprfs_work*, *LAPACKE_ssprfs_work* and *LAPACKE_zsprfs_work*.

4.19.1033 LAPACKE_dspsv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dspsv(armpl_int_t matrix_layout, char uplo, armpl_int_t n,
                          armpl_int_t nrhs, double *ap, armpl_int_t *ipiv,
                          double *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cspsv*, *LAPACKE_dspsv*, *LAPACKE_sspsv* and *LAPACKE_zspsv*. It also exists with a native Fortran interface as *dspsv*.

4.19.1034 LAPACKE_dspsv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dspsv_work(armpl_int_t matrix_layout, char uplo,
                               armpl_int_t n, armpl_int_t nrhs, double *ap,
                               armpl_int_t *ipiv, double *b,
                               armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cspsv_work*, *LAPACKE_dspsv_work*, *LAPACKE_sspsv_work* and *LAPACKE_zspsv_work*.

4.19.1035 LAPACKE_dspsvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dspsvx(armpl_int_t matrix_layout, char fact, char uplo,
                           armpl_int_t n, armpl_int_t nrhs, const double *ap,
                           double *afp, armpl_int_t *ipiv, const double *b,
                           armpl_int_t ldb, double *x, armpl_int_t ldx,
                           double *rcond, double *ferr, double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cspsvx*, *LAPACKE_dspsvx*, *LAPACKE_sspsvx* and *LAPACKE_zspsvx*. It also exists with a native Fortran interface as *dspsvx*.

4.19.1036 LAPACKE_dspsvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dspsvx_work(armpl_int_t matrix_layout, char fact,
                                char uplo, armpl_int_t n, armpl_int_t nrhs,
                                const double *ap, double *afp,
                                armpl_int_t *ipiv, const double *b,
                                armpl_int_t ldb, double *x, armpl_int_t ldx,
                                double *rcond, double *ferr, double *berr,
                                double *work, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cspsvx_work*, *LAPACKE_dspsvx_work*, *LAPACKE_sspsvx_work* and *LAPACKE_zspsvx_work*.

4.19.1037 LAPACKE_dsptrd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsptrd(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, double *ap, double *d, double *e,
                           double *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsptrd* and *LAPACKE_ssptrd*. It also exists with a native Fortran interface as *dsptrd*.

4.19.1038 LAPACKE_dsptrd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsptrd_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, double *ap, double *d,
                                double *e, double *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsptrd_work* and *LAPACKE_ssptrd_work*.

4.19.1039 LAPACKE_dsptrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsptrf(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, double *ap, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csptrf*, *LAPACKE_dsptrf*, *LAPACKE_ssptrf* and *LAPACKE_zsptrf*. It also exists with a native Fortran interface as *dsptrf*.

4.19.1040 LAPACKE_dsptrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsptrf_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, double *ap,
                                armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csptrf_work*, *LAPACKE_dsptrf_work*, *LAPACKE_ssptrf_work* and *LAPACKE_zsptrf_work*.

4.19.1041 LAPACKE_dsptri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsptri(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, double *ap,
                           const armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csptri*, *LAPACKE_dsptri*, *LAPACKE_ssptri* and *LAPACKE_zsptri*. It also exists with a native Fortran interface as *dsptri*.

4.19.1042 LAPACKE_dsptri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsptri_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, double *ap,
                                const armpl_int_t *ipiv, double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csptri_work*, *LAPACKE_dsptri_work*, *LAPACKE_ssptri_work* and *LAPACKE_zsptri_work*.

4.19.1043 LAPACKE_dsptrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsptrs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs, const double *ap,
                           const armpl_int_t *ipiv, double *b,
                           armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csptrs*, *LAPACKE_dsptrs*, *LAPACKE_ssptrs* and *LAPACKE_zsptrs*. It also exists with a native Fortran interface as *dsptrs*.

4.19.1044 LAPACKE_dsptrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsptrs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const double *ap, const armpl_int_t *ipiv,
                                double *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csptrs_work*, *LAPACKE_dsptrs_work*, *LAPACKE_ssptrs_work* and *LAPACKE_zsptrs_work*.

4.19.1045 LAPACKE_dstebz

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dstebz(char range, char order, armpl_int_t n, double vl,
                           double vu, armpl_int_t il, armpl_int_t iu,
                           double abstol, const double *d, const double *e,
                           armpl_int_t *m, armpl_int_t *nsplit, double *w,
                           armpl_int_t *iblock, armpl_int_t *isplit);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dstebz* and *LAPACKE_sstebz*. It also exists with a native Fortran interface as *dstebz*.

4.19.1046 LAPACKE_dstebz_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dstebz_work(char range, char order, armpl_int_t n,
                                double vl, double vu, armpl_int_t il,
                                armpl_int_t iu, double abstol,
                                const double *d, const double *e,
                                armpl_int_t *m, armpl_int_t *nsplit,
                                double *w, armpl_int_t *iblock,
                                armpl_int_t *isplit, double *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dstebz_work* and *LAPACKE_sstebz_work*.

4.19.1047 LAPACKE_dstedc

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dstedc(armpl_int_t matrix_layout, char compz,
                           armpl_int_t n, double *d, double *e, double *z,
                           armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cstedc*, *LAPACKE_dstedc*, *LAPACKE_sstedc* and *LAPACKE_zstedc*. It also exists with a native Fortran interface as *dstedc*.

4.19.1048 LAPACKE_dstedc_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dstedc_work(armpl_int_t matrix_layout, char compz,
                                armpl_int_t n, double *d, double *e,
                                double *z, armpl_int_t ldz, double *work,
                                armpl_int_t lwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cstedc_work*, *LAPACKE_dstedc_work*, *LAPACKE_sstedc_work* and *LAPACKE_zstedc_work*.

4.19.1049 LAPACKE_dstegr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dstegr(armpl_int_t matrix_layout, char jobz, char range,
                           armpl_int_t n, double *d, double *e, double vl,
                           double vu, armpl_int_t il, armpl_int_t iu,
                           double abstol, armpl_int_t *m, double *w,
                           double *z, armpl_int_t ldz, armpl_int_t *isuppz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cstegr*, *LAPACKE_dstegr*, *LAPACKE_sstegr* and *LAPACKE_zstegr*. It also exists with a native Fortran interface as *dstegr*.

4.19.1050 LAPACKE_dstegr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dstegr_work(armpl_int_t matrix_layout, char jobz,
                                char range, armpl_int_t n, double *d,
                                double *e, double vl, double vu,
                                armpl_int_t il, armpl_int_t iu, double abstol,
                                armpl_int_t *m, double *w, double *z,
                                armpl_int_t ldz, armpl_int_t *isuppz,
                                double *work, armpl_int_t lwork,
                                armpl_int_t *iwork, armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cstegr_work*, *LAPACKE_dstegr_work*, *LAPACKE_sstegr_work* and *LAPACKE_zstegr_work*.

4.19.1051 LAPACKE_dstein

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dstein(armpl_int_t matrix_layout, armpl_int_t n,
                           const double *d, const double *e, armpl_int_t m,
                           const double *w, const armpl_int_t *iblock,
                           const armpl_int_t *isplit, double *z,
                           armpl_int_t ldz, armpl_int_t *ifailv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cstein*, *LAPACKE_dstein*, *LAPACKE_sstein* and *LAPACKE_zstein*. It also exists with a native Fortran interface as *dstein*.

4.19.1052 LAPACKE_dstein_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dstein_work(armpl_int_t matrix_layout, armpl_int_t n,
                                const double *d, const double *e,
                                armpl_int_t m, const double *w,
                                const armpl_int_t *iblock,
                                const armpl_int_t *isplit, double *z,
                                armpl_int_t ldz, double *work,
                                armpl_int_t *iwork, armpl_int_t *ifailv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cstein_work*, *LAPACKE_dstein_work*, *LAPACKE_sstein_work* and *LAPACKE_zstein_work*.

4.19.1053 LAPACKE_dstemr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dstemr(armpl_int_t matrix_layout, char jobz, char range,
                           armpl_int_t n, double *d, double *e, double vl,
                           double vu, armpl_int_t il, armpl_int_t iu,
                           armpl_int_t *m, double *w, double *z,
                           armpl_int_t ldz, armpl_int_t nzc,
                           armpl_int_t *isuppz, armpl_int_t *tryrac);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cstemr*, *LAPACKE_dstemr*, *LAPACKE_sstemr* and *LAPACKE_zstemr*. It also exists with a native Fortran interface as *dstemr*.

4.19.1054 LAPACKE_dstemr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dstemr_work(armpl_int_t matrix_layout, char jobz,
                                char range, armpl_int_t n, double *d,
                                double *e, double vl, double vu,
                                armpl_int_t il, armpl_int_t iu,
                                armpl_int_t *m, double *w, double *z,
                                armpl_int_t ldz, armpl_int_t nzc,
                                armpl_int_t *isuppz, armpl_int_t *tryrac,
                                double *work, armpl_int_t lwork,
                                armpl_int_t *iwork, armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cstemr_work*, *LAPACKE_dstemr_work*, *LAPACKE_sstemr_work* and *LAPACKE_zstemr_work*.

4.19.1055 LAPACKE_dsteqr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsteqr(armpl_int_t matrix_layout, char compz,
                           armpl_int_t n, double *d, double *e, double *z,
                           armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csteqr*, *LAPACKE_dsteqr*, *LAPACKE_ssteqr* and *LAPACKE_zsteqr*. It also exists with a native Fortran interface as *dsteqr*.

4.19.1056 LAPACKE_dsteqr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsteqr_work(armpl_int_t matrix_layout, char compz,
                                armpl_int_t n, double *d, double *e,
                                double *z, armpl_int_t ldz, double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.1057 LAPACKE_dsterf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsterf(armpl_int_t n, double *d, double *e);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsterf* and *LAPACKE_ssterf*. It also exists with a native Fortran interface as *dsterf*.

4.19.1058 LAPACKE_dsterf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsterf_work(armpl_int_t n, double *d, double *e);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsterf_work* and *LAPACKE_ssterf_work*.

4.19.1059 LAPACKE_dstev

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dstev(armpl_int_t matrix_layout, char jobz, armpl_int_t n,
                          double *d, double *e, double *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dstev* and *LAPACKE_sstev*. It also exists with a native Fortran interface as *dstev*.

4.19.1060 LAPACKE_dstev_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dstev_work(armpl_int_t matrix_layout, char jobz,
                               armpl_int_t n, double *d, double *e, double *z,
                               armpl_int_t ldz, double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dstev_work* and *LAPACKE_sstev_work*.

4.19.1061 LAPACKE_dstevd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dstevd(armpl_int_t matrix_layout, char jobz,
                           armpl_int_t n, double *d, double *e, double *z,
                           armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dstevd* and *LAPACKE_sstevd*. It also exists with a native Fortran interface as *dstevd*.

4.19.1062 LAPACKE_dstevd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dstevd_work(armpl_int_t matrix_layout, char jobz,
                                armpl_int_t n, double *d, double *e,
                                double *z, armpl_int_t ldz, double *work,
                                armpl_int_t lwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dstevd_work* and *LAPACKE_sstevd_work*.

4.19.1063 LAPACKE_dstevr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dstevr(armpl_int_t matrix_layout, char jobz, char range,
                           armpl_int_t n, double *d, double *e, double vl,
                           double vu, armpl_int_t il, armpl_int_t iu,
                           double abstol, armpl_int_t *m, double *w,
                           double *z, armpl_int_t ldz, armpl_int_t *isuppz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dstevr* and *LAPACKE_sstevr*. It also exists with a native Fortran interface as *dstevr*.

4.19.1064 LAPACKE_dstevr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dstevr_work(armpl_int_t matrix_layout, char jobz,
                                char range, armpl_int_t n, double *d,
                                double *e, double vl, double vu,
                                armpl_int_t il, armpl_int_t iu, double abstol,
                                armpl_int_t *m, double *w, double *z,
                                armpl_int_t ldz, armpl_int_t *isuppz,
                                double *work, armpl_int_t lwork,
                                armpl_int_t *iwork, armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dstevr_work* and *LAPACKE_sstevr_work*.

4.19.1065 LAPACKE_dstevx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dstevx(armpl_int_t matrix_layout, char jobz, char range,
                           armpl_int_t n, double *d, double *e, double vl,
                           double vu, armpl_int_t il, armpl_int_t iu,
                           double abstol, armpl_int_t *m, double *w,
                           double *z, armpl_int_t ldz, armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dstevx* and *LAPACKE_sstevx*. It also exists with a native Fortran interface as *dstevx*.

4.19.1066 LAPACKE_dstevx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dstevx_work(armpl_int_t matrix_layout, char jobz,
                                char range, armpl_int_t n, double *d,
                                double *e, double vl, double vu,
                                armpl_int_t il, armpl_int_t iu, double abstol,
                                armpl_int_t *m, double *w, double *z,
                                armpl_int_t ldz, double *work,
                                armpl_int_t *iwork, armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dstevx_work* and *LAPACKE_sstevx_work*.

4.19.1067 LAPACKE_dsycon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsycon(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, const double *a, armpl_int_t lda,
                           const armpl_int_t *ipiv, double anorm,
                           double *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csycon*, *LAPACKE_dsycon*, *LAPACKE_ssycon* and *LAPACKE_zsycon*. It also exists with a native Fortran interface as *dsycon*.

4.19.1068 LAPACKE_dsycon_3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsycon_3(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, const double *a, armpl_int_t lda,
                             const double *e, const armpl_int_t *ipiv,
                             double anorm, double *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csycon_3*, *LAPACKE_dsycon_3*, *LAPACKE_ssycon_3* and *LAPACKE_zsycon_3*. It also exists with a native Fortran interface as *dsycon_3*.

4.19.1069 LAPACKE_dsycon_3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsycon_3_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, const double *a,
                                  armpl_int_t lda, const double *e,
                                  const armpl_int_t *ipiv, double anorm,
                                  double *rcond, double *work,
                                  armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csycon_3_work*, *LAPACKE_dsycon_3_work*, *LAPACKE_ssycon_3_work* and *LAPACKE_zsycon_3_work*.

4.19.1070 LAPACKE_dsycon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsycon_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, const double *a,
                                armpl_int_t lda, const armpl_int_t *ipiv,
                                double anorm, double *rcond, double *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csycon_work*, *LAPACKE_dsycon_work*, *LAPACKE_ssycon_work* and *LAPACKE_zsycon_work*.

4.19.1071 LAPACKE_dsyconv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsyconv(armpl_int_t matrix_layout, char uplo, char way,
                            armpl_int_t n, double *a, armpl_int_t lda,
                            const armpl_int_t *ipiv, double *e);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyconv*, *LAPACKE_dsyconv*, *LAPACKE_ssyconv* and *LAPACKE_zsyconv*. It also exists with a native Fortran interface as *dsyconv*.

4.19.1072 LAPACKE_dsyconv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsyconv_work(armpl_int_t matrix_layout, char uplo,
                                 char way, armpl_int_t n, double *a,
                                 armpl_int_t lda, const armpl_int_t *ipiv,
                                 double *e);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyconv_work*, *LAPACKE_dsyconv_work*, *LAPACKE_ssyconv_work* and *LAPACKE_zsyconv_work*.

4.19.1073 LAPACKE_dsyequb

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsyequb(armpl_int_t matrix_layout, char uplo,
                            armpl_int_t n, const double *a, armpl_int_t lda,
                            double *s, double *scond, double *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyequb*, *LAPACKE_dsyequb*, *LAPACKE_ssyequb* and *LAPACKE_zsyequb*. It also exists with a native Fortran interface as *dsyequb*.

4.19.1074 LAPACKE_dsyequb_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsyequb_work(armpl_int_t matrix_layout, char uplo,
                                 armpl_int_t n, const double *a,
                                 armpl_int_t lda, double *s, double *scond,
                                 double *amax, double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyequb_work*, *LAPACKE_dsyequb_work*, *LAPACKE_ssyequb_work* and *LAPACKE_zsyequb_work*.

4.19.1075 LAPACKE_dsyev

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsyev(armpl_int_t matrix_layout, char jobz, char uplo,
                          armpl_int_t n, double *a, armpl_int_t lda,
                          double *w);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsyev* and *LAPACKE_ssyev*. It also exists with a native Fortran interface as *dsyev*.

4.19.1076 LAPACKE_dsyev_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsyev_2stage(armpl_int_t matrix_layout, char jobz,
                                 char uplo, armpl_int_t n, double *a,
                                 armpl_int_t lda, double *w);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsyev_2stage* and *LAPACKE_ssyev_2stage*. It also exists with a native Fortran interface as *dsyev_2stage*.

4.19.1077 LAPACKE_dsyev_2stage_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsyev_2stage_work(armpl_int_t matrix_layout, char jobz,
                                      char uplo, armpl_int_t n, double *a,
                                      armpl_int_t lda, double *w,
                                      double *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsyev_2stage_work* and *LAPACKE_ssyev_2stage_work*.

4.19.1078 LAPACKE_dsyev_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsyev_work(armpl_int_t matrix_layout, char jobz,
                               char uplo, armpl_int_t n, double *a,
                               armpl_int_t lda, double *w, double *work,
                               armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsyev_work* and *LAPACKE_ssyev_work*.

4.19.1079 LAPACKE_dsyevd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsyevd(armpl_int_t matrix_layout, char jobz, char uplo,
                           armpl_int_t n, double *a, armpl_int_t lda,
                           double *w);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsyevd* and *LAPACKE_ssyevd*. It also exists with a native Fortran interface as *dsyevd*.

4.19.1080 LAPACKE_dsyevd_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsyevd_2stage(armpl_int_t matrix_layout, char jobz,
                                  char uplo, armpl_int_t n, double *a,
                                  armpl_int_t lda, double *w);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsyevd_2stage* and *LAPACKE_ssyevd_2stage*. It also exists with a native Fortran interface as *dsyevd_2stage*.

4.19.1081 LAPACKE_dsyevd_2stage_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsyevd_2stage_work(armpl_int_t matrix_layout, char jobz,
                                       char uplo, armpl_int_t n, double *a,
                                       armpl_int_t lda, double *w,
                                       double *work, armpl_int_t lwork,
                                       armpl_int_t *iwork,
                                       armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsyevd_2stage_work* and *LAPACKE_ssyevd_2stage_work*.

4.19.1082 LAPACKE_dsyevd_work

4.19.1083 LAPACKE_dsyevr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsyevr(armpl_int_t matrix_layout, char jobz, char range,
                           char uplo, armpl_int_t n, double *a,
                           armpl_int_t lda, double vl, double vu,
                           armpl_int_t il, armpl_int_t iu, double abstol,
                           armpl_int_t *m, double *w, double *z,
                           armpl_int_t ldz, armpl_int_t *isuppz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsyevr* and *LAPACKE_ssyevr*. It also exists with a native Fortran interface as *dsyevr*.

4.19.1084 LAPACKE_dsyevr_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsyevr_2stage(armpl_int_t matrix_layout, char jobz,
                                  char range, char uplo, armpl_int_t n,
                                  double *a, armpl_int_t lda, double vl,
                                  double vu, armpl_int_t il, armpl_int_t iu,
                                  double abstol, armpl_int_t *m, double *w,
                                  double *z, armpl_int_t ldz,
                                  armpl_int_t *isuppz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsyevr_2stage* and *LAPACKE_ssyevr_2stage*. It also exists with a native Fortran interface as *dsyevr_2stage*.

4.19.1085 LAPACKE_dsyevr_2stage_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsyevr_2stage_work(armpl_int_t matrix_layout, char jobz,
                                       char range, char uplo, armpl_int_t n,
                                       double *a, armpl_int_t lda, double vl,
                                       double vu, armpl_int_t il,
                                       armpl_int_t iu, double abstol,
                                       armpl_int_t *m, double *w, double *z,
                                       armpl_int_t ldz, armpl_int_t *isuppz,
                                       double *work, armpl_int_t lwork,
                                       armpl_int_t *iwork,
                                       armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsyevr_2stage_work* and *LAPACKE_ssyevr_2stage_work*.

4.19.1086 LAPACKE_dsyevr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsyevr_work(armpl_int_t matrix_layout, char jobz,
                                char range, char uplo, armpl_int_t n,
                                double *a, armpl_int_t lda, double vl,
                                double vu, armpl_int_t il, armpl_int_t iu,
                                double abstol, armpl_int_t *m, double *w,
                                double *z, armpl_int_t ldz,
                                armpl_int_t *isuppz, double *work,
                                armpl_int_t lwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsyevr_work* and *LAPACKE_ssyevr_work*.

4.19.1087 LAPACKE_dsyevx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsyevx(armpl_int_t matrix_layout, char jobz, char range,
                           char uplo, armpl_int_t n, double *a,
                           armpl_int_t lda, double vl, double vu,
                           armpl_int_t il, armpl_int_t iu, double abstol,
                           armpl_int_t *m, double *w, double *z,
                           armpl_int_t ldz, armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsyevx* and *LAPACKE_ssyevx*. It also exists with a native Fortran interface as *dsyevx*.

4.19.1088 LAPACKE_dsyevx_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsyevx_2stage(armpl_int_t matrix_layout, char jobz,
                                  char range, char uplo, armpl_int_t n,
                                  double *a, armpl_int_t lda, double vl,
                                  double vu, armpl_int_t il, armpl_int_t iu,
                                  double abstol, armpl_int_t *m, double *w,
                                  double *z, armpl_int_t ldz,
                                  armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsyevx_2stage* and *LAPACKE_ssyevx_2stage*. It also exists with a native Fortran interface as *dsyevx_2stage*.

4.19.1089 LAPACKE_dsyevx_2stage_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsyevx_2stage_work(armpl_int_t matrix_layout, char jobz,
                                       char range, char uplo, armpl_int_t n,
                                       double *a, armpl_int_t lda, double vl,
                                       double vu, armpl_int_t il,
                                       armpl_int_t iu, double abstol,
                                       armpl_int_t *m, double *w, double *z,
                                       armpl_int_t ldz, double *work,
                                       armpl_int_t lwork, armpl_int_t *iwork,
                                       armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsyevx_2stage_work* and *LAPACKE_ssyevx_2stage_work*.

4.19.1090 LAPACKE_dsyevx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsyevx_work(armpl_int_t matrix_layout, char jobz,
                                char range, char uplo, armpl_int_t n,
                                double *a, armpl_int_t lda, double vl,
                                double vu, armpl_int_t il, armpl_int_t iu,
                                double abstol, armpl_int_t *m, double *w,
                                double *z, armpl_int_t ldz, double *work,
                                armpl_int_t lwork, armpl_int_t *iwork,
                                armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsyevx_work* and *LAPACKE_ssyevx_work*.

4.19.1091 LAPACKE_dsygst

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsygst(armpl_int_t matrix_layout, armpl_int_t itype,
                           char uplo, armpl_int_t n, double *a,
                           armpl_int_t lda, const double *b,
                           armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsygst* and *LAPACKE_ssygst*. It also exists with a native Fortran interface as *dsygst*.

4.19.1092 LAPACKE_dsygst_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsygst_work(armpl_int_t matrix_layout, armpl_int_t itype,
                                char uplo, armpl_int_t n, double *a,
                                armpl_int_t lda, const double *b,
                                armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsygst_work* and *LAPACKE_ssygst_work*.

4.19.1093 LAPACKE_dsygv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsygv(armpl_int_t matrix_layout, armpl_int_t itype,
                          char jobz, char uplo, armpl_int_t n, double *a,
                          armpl_int_t lda, double *b, armpl_int_t ldb,
                          double *w);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsygv* and *LAPACKE_ssygv*. It also exists with a native Fortran interface as *dsygv*.

4.19.1094 LAPACKE_dsygv_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsygv_2stage(armpl_int_t matrix_layout, armpl_int_t itype,
                                 char jobz, char uplo, armpl_int_t n,
                                 double *a, armpl_int_t lda, double *b,
                                 armpl_int_t ldb, double *w);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsygv_2stage* and *LAPACKE_ssygv_2stage*. It also exists with a native Fortran interface as *dsygv_2stage*.

4.19.1095 LAPACKE_dsygv_2stage_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsygv_2stage_work(armpl_int_t matrix_layout,
                                      armpl_int_t itype, char jobz, char uplo,
                                      armpl_int_t n, double *a,
                                      armpl_int_t lda, double *b,
                                      armpl_int_t ldb, double *w,
                                      double *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsygv_2stage_work* and *LAPACKE_ssygv_2stage_work*.

4.19.1096 LAPACKE_dsygv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsygv_work(armpl_int_t matrix_layout, armpl_int_t itype,
                               char jobz, char uplo, armpl_int_t n, double *a,
                               armpl_int_t lda, double *b, armpl_int_t ldb,
                               double *w, double *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsygv_work* and *LAPACKE_ssygv_work*.

4.19.1097 LAPACKE_dsygvd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsygvd(armpl_int_t matrix_layout, armpl_int_t itype,
                           char jobz, char uplo, armpl_int_t n, double *a,
                           armpl_int_t lda, double *b, armpl_int_t ldb,
                           double *w);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsygvd* and *LAPACKE_ssygvd*. It also exists with a native Fortran interface as *dsygvd*.

4.19.1098 LAPACKE_dsygvd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsygvd_work(armpl_int_t matrix_layout, armpl_int_t itype,
                                char jobz, char uplo, armpl_int_t n,
                                double *a, armpl_int_t lda, double *b,
                                armpl_int_t ldb, double *w, double *work,
                                armpl_int_t lwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsygvd_work* and *LAPACKE_ssygvd_work*.

4.19.1099 LAPACKE_dsygvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsygvx(armpl_int_t matrix_layout, armpl_int_t itype,
                           char jobz, char range, char uplo, armpl_int_t n,
                           double *a, armpl_int_t lda, double *b,
                           armpl_int_t ldb, double vl, double vu,
```

(continues on next page)

(continued from previous page)

```
                                armpl_int_t il, armpl_int_t iu, double abstol,
                                armpl_int_t *m, double *w, double *z,
                                armpl_int_t ldz, armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsygvx* and *LAPACKE_ssygvx*. It also exists with a native Fortran interface as *dsygvx*.

4.19.1100 LAPACKE_dsygvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsygvx_work(armpl_int_t matrix_layout, armpl_int_t itype,
                                char jobz, char range, char uplo,
                                armpl_int_t n, double *a, armpl_int_t lda,
                                double *b, armpl_int_t ldb, double vl,
                                double vu, armpl_int_t il, armpl_int_t iu,
                                double abstol, armpl_int_t *m, double *w,
                                double *z, armpl_int_t ldz, double *work,
                                armpl_int_t lwork, armpl_int_t *iwork,
                                armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsygvx_work* and *LAPACKE_ssygvx_work*.

4.19.1101 LAPACKE_dsyrfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsyrfs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs, const double *a,
                           armpl_int_t lda, const double *af,
                           armpl_int_t ldaf, const armpl_int_t *ipiv,
                           const double *b, armpl_int_t ldb, double *x,
                           armpl_int_t ldx, double *ferr, double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyrfs*, *LAPACKE_dsyrfs*, *LAPACKE_ssyrfs* and *LAPACKE_zsyrfs*. It also exists with a native Fortran interface as *dsyrfs*.

4.19.1102 LAPACKE_dsyrfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsyrfs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const double *a, armpl_int_t lda,
                                const double *af, armpl_int_t ldaf,
                                const armpl_int_t *ipiv, const double *b,
                                armpl_int_t ldb, double *x, armpl_int_t ldx,
                                double *ferr, double *berr, double *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyrfs_work*, *LAPACKE_dsyrfs_work*, *LAPACKE_ssyrfs_work* and *LAPACKE_zsyrfs_work*.

4.19.1103 LAPACKE_dsyrfsx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsyrfsx(armpl_int_t matrix_layout, char uplo, char equed,
                            armpl_int_t n, armpl_int_t nrhs, const double *a,
                            armpl_int_t lda, const double *af,
                            armpl_int_t ldaf, const armpl_int_t *ipiv,
                            const double *s, const double *b, armpl_int_t ldb,
                            double *x, armpl_int_t ldx, double *rcond,
                            double *berr, armpl_int_t n_err_bnds,
                            double *err_bnds_norm, double *err_bnds_comp,
                            armpl_int_t nparams, double *params);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyrfsx*, *LAPACKE_dsyrfsx*, *LAPACKE_ssyrfsx* and *LAPACKE_zsyrfsx*. It also exists with a native Fortran interface as *dsyrfsx*.

4.19.1104 LAPACKE_dsyrfsx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsyrfsx_work(armpl_int_t matrix_layout, char uplo,
                                 char equed, armpl_int_t n, armpl_int_t nrhs,
                                 const double *a, armpl_int_t lda,
                                 const double *af, armpl_int_t ldaf,
                                 const armpl_int_t *ipiv, const double *s,
                                 const double *b, armpl_int_t ldb, double *x,
                                 armpl_int_t ldx, double *rcond, double *berr,
                                 armpl_int_t n_err_bnds,
                                 double *err_bnds_norm, double *err_bnds_comp,
                                 armpl_int_t nparams, double *params,
                                 double *work, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyrfsx_work*, *LAPACKE_dsyrfsx_work*, *LAPACKE_ssyrfsx_work* and *LAPACKE_zsyrfsx_work*.

4.19.1105 LAPACKE_dsysv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsysv(armpl_int_t matrix_layout, char uplo, armpl_int_t n,
                          armpl_int_t nrhs, double *a, armpl_int_t lda,
                          armpl_int_t *ipiv, double *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysv*, *LAPACKE_dsysv*, *LAPACKE_ssysv* and *LAPACKE_zsysv*. It also exists with a native Fortran interface as *dsysv*.

4.19.1106 LAPACKE_dsysv_aa

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsysv_aa(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, armpl_int_t nrhs, double *a,
                             armpl_int_t lda, armpl_int_t *ipiv, double *b,
                             armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysv_aa*, *LAPACKE_dsysv_aa*, *LAPACKE_ssysv_aa* and *LAPACKE_zsysv_aa*. It also exists with a native Fortran interface as *dsysv_aa*.

4.19.1107 LAPACKE_dsysv_aa_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsysv_aa_2stage(armpl_int_t matrix_layout, char uplo,
                                    armpl_int_t n, armpl_int_t nrhs,
                                    double *a, armpl_int_t lda, double *tb,
                                    armpl_int_t ltb, armpl_int_t *ipiv,
                                    armpl_int_t *ipiv2, double *b,
                                    armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysv_aa_2stage*, *LAPACKE_dsysv_aa_2stage*, *LAPACKE_ssysv_aa_2stage* and *LAPACKE_zsysv_aa_2stage*. It also exists with a native Fortran interface as *dsysv_aa_2stage*.

4.19.1108 LAPACKE_dsysv_aa_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsysv_aa_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, armpl_int_t nrhs, double *a,
```

(continues on next page)

(continued from previous page)

```
                                armpl_int_t lda, armpl_int_t *ipiv,
                                double *b, armpl_int_t ldb, double *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysv_aa_work*, *LAPACKE_dsysv_aa_work*, *LAPACKE_ssysv_aa_work* and *LAPACKE_zsysv_aa_work*.

4.19.1109 LAPACKE_dsysv_rk

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsysv_rk(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, armpl_int_t nrhs, double *a,
                             armpl_int_t lda, double *e, armpl_int_t *ipiv,
                             double *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysv_rk*, *LAPACKE_dsysv_rk*, *LAPACKE_ssysv_rk* and *LAPACKE_zsysv_rk*. It also exists with a native Fortran interface as *dsysv_rk*.

4.19.1110 LAPACKE_dsysv_rk_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsysv_rk_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, armpl_int_t nrhs, double *a,
                                  armpl_int_t lda, double *e,
                                  armpl_int_t *ipiv, double *b,
                                  armpl_int_t ldb, double *work,
                                  armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysv_rk_work*, *LAPACKE_dsysv_rk_work*, *LAPACKE_ssysv_rk_work* and *LAPACKE_zsysv_rk_work*.

4.19.1111 LAPACKE_dsysv_rook

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsysv_rook(armpl_int_t matrix_layout, char uplo,
                               armpl_int_t n, armpl_int_t nrhs, double *a,
                               armpl_int_t lda, armpl_int_t *ipiv, double *b,
                               armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysv_rook*, *LAPACKE_dsysv_rook*, *LAPACKE_ssysv_rook* and *LAPACKE_zsysv_rook*. It also exists with a native Fortran interface as *dsysv_rook*.

4.19.1112 LAPACKE_dsysv_rook_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsysv_rook_work(armpl_int_t matrix_layout, char uplo,
                                    armpl_int_t n, armpl_int_t nrhs,
                                    double *a, armpl_int_t lda,
                                    armpl_int_t *ipiv, double *b,
                                    armpl_int_t ldb, double *work,
                                    armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysv_rook_work*, *LAPACKE_dsysv_rook_work*, *LAPACKE_ssysv_rook_work* and *LAPACKE_zsysv_rook_work*.

4.19.1113 LAPACKE_dsysv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsysv_work(armpl_int_t matrix_layout, char uplo,
                               armpl_int_t n, armpl_int_t nrhs, double *a,
                               armpl_int_t lda, armpl_int_t *ipiv, double *b,
                               armpl_int_t ldb, double *work,
                               armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysv_work*, *LAPACKE_dsysv_work*, *LAPACKE_ssysv_work* and *LAPACKE_zsysv_work*.

4.19.1114 LAPACKE_dsysvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsysvx(armpl_int_t matrix_layout, char fact, char uplo,
                           armpl_int_t n, armpl_int_t nrhs, const double *a,
                           armpl_int_t lda, double *af, armpl_int_t ldaf,
                           armpl_int_t *ipiv, const double *b,
                           armpl_int_t ldb, double *x, armpl_int_t ldx,
                           double *rcond, double *ferr, double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysvx*, *LAPACKE_dsysvx*, *LAPACKE_ssysvx* and *LAPACKE_zsysvx*. It also exists with a native Fortran interface as *dsysvx*.

4.19.1115 LAPACKE_dsysvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsysvx_work(armpl_int_t matrix_layout, char fact,
                                char uplo, armpl_int_t n, armpl_int_t nrhs,
                                const double *a, armpl_int_t lda, double *af,
                                armpl_int_t ldaf, armpl_int_t *ipiv,
                                const double *b, armpl_int_t ldb, double *x,
                                armpl_int_t ldx, double *rcond, double *ferr,
                                double *berr, double *work, armpl_int_t lwork,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysvx_work*, *LAPACKE_dsysvx_work*, *LAPACKE_ssysvx_work* and *LAPACKE_zsysvx_work*.

4.19.1116 LAPACKE_dsysvxx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsysvxx(armpl_int_t matrix_layout, char fact, char uplo,
                            armpl_int_t n, armpl_int_t nrhs, double *a,
                            armpl_int_t lda, double *af, armpl_int_t ldaf,
                            armpl_int_t *ipiv, char *equed, double *s,
                            double *b, armpl_int_t ldb, double *x,
                            armpl_int_t ldx, double *rcond, double *rpvgrw,
                            double *berr, armpl_int_t n_err_bnds,
                            double *err_bnds_norm, double *err_bnds_comp,
                            armpl_int_t nparams, double *params);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysvxx*, *LAPACKE_dsysvxx*, *LAPACKE_ssysvxx* and *LAPACKE_zsysvxx*. It also exists with a native Fortran interface as *dsysvxx*.

4.19.1117 LAPACKE_dsysvxx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsysvxx_work(armpl_int_t matrix_layout, char fact,
                                 char uplo, armpl_int_t n, armpl_int_t nrhs,
                                 double *a, armpl_int_t lda, double *af,
                                 armpl_int_t ldaf, armpl_int_t *ipiv,
                                 char *equed, double *s, double *b,
                                 armpl_int_t ldb, double *x, armpl_int_t ldx,
```

(continues on next page)

(continued from previous page)

```
                                double *rcond, double *rpvgrw, double *berr,
                                armpl_int_t n_err_bnds,
                                double *err_bnds_norm, double *err_bnds_comp,
                                armpl_int_t nparams, double *params,
                                double *work, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysvxx_work*, *LAPACKE_dsysvxx_work*, *LAPACKE_ssysvxx_work* and *LAPACKE_zsysvxx_work*.

4.19.1118 LAPACKE_dsyswapr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsyswapr(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, double *a, armpl_int_t lda,
                             armpl_int_t i1, armpl_int_t i2);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyswapr*, *LAPACKE_dsyswapr*, *LAPACKE_ssyswapr* and *LAPACKE_zsyswapr*. It also exists with a native Fortran interface as *dsyswapr*.

4.19.1119 LAPACKE_dsyswapr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsyswapr_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, double *a, armpl_int_t lda,
                                  armpl_int_t i1, armpl_int_t i2);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyswapr_work*, *LAPACKE_dsyswapr_work*, *LAPACKE_ssyswapr_work* and *LAPACKE_zsyswapr_work*.

4.19.1120 LAPACKE_dsytrd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsytrd(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, double *a, armpl_int_t lda,
                           double *d, double *e, double *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsytrd* and *LAPACKE_ssytrd*. It also exists with a native Fortran interface as *dsytrd*.

4.19.1121 LAPACKE_dsytrd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsytrd_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, double *a, armpl_int_t lda,
                                double *d, double *e, double *tau,
                                double *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsytrd_work* and *LAPACKE_ssytrd_work*.

4.19.1122 LAPACKE_dsytrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsytrf(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, double *a, armpl_int_t lda,
                           armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrf*, *LAPACKE_dsytrf*, *LAPACKE_ssytrf* and *LAPACKE_zsytrf*. It also exists with a native Fortran interface as *dsytrf*.

4.19.1123 LAPACKE_dsytrf_aa

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsytrf_aa(armpl_int_t matrix_layout, char uplo,
                              armpl_int_t n, double *a, armpl_int_t lda,
                              armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrf_aa*, *LAPACKE_dsytrf_aa*, *LAPACKE_ssytrf_aa* and *LAPACKE_zsytrf_aa*. It also exists with a native Fortran interface as *dsytrf_aa*.

4.19.1124 LAPACKE_dsytrf_aa_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsytrf_aa_2stage(armpl_int_t matrix_layout, char uplo,
                                     armpl_int_t n, double *a,
                                     armpl_int_t lda, double *tb,
                                     armpl_int_t ltb, armpl_int_t *ipiv,
                                     armpl_int_t *ipiv2);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrf_aa_2stage*, *LAPACKE_dsytrf_aa_2stage*, *LAPACKE_ssytrf_aa_2stage* and *LAPACKE_zsytrf_aa_2stage*. It also exists with a native Fortran interface as *dsytrf_aa_2stage*.

4.19.1125 LAPACKE_dsytrf_aa_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsytrf_aa_work(armpl_int_t matrix_layout, char uplo,
                                   armpl_int_t n, double *a, armpl_int_t lda,
                                   armpl_int_t *ipiv, double *work,
                                   armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrf_aa_work*, *LAPACKE_dsytrf_aa_work*, *LAPACKE_ssytrf_aa_work* and *LAPACKE_zsytrf_aa_work*.

4.19.1126 LAPACKE_dsytrf_rk

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsytrf_rk(armpl_int_t matrix_layout, char uplo,
                              armpl_int_t n, double *a, armpl_int_t lda,
                              double *e, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrf_rk*, *LAPACKE_dsytrf_rk*, *LAPACKE_ssytrf_rk* and *LAPACKE_zsytrf_rk*. It also exists with a native Fortran interface as *dsytrf_rk*.

4.19.1127 LAPACKE_dsytrf_rk_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsytrf_rk_work(armpl_int_t matrix_layout, char uplo,
                                   armpl_int_t n, double *a, armpl_int_t lda,
                                   double *e, armpl_int_t *ipiv, double *work,
                                   armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrf_rk_work*, *LAPACKE_dsytrf_rk_work*, *LAPACKE_ssytrf_rk_work* and *LAPACKE_zsytrf_rk_work*.

4.19.1128 LAPACKE_dsytrf_rook

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsytrf_rook(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, double *a, armpl_int_t lda,
                                armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrf_rook*, *LAPACKE_dsytrf_rook*, *LAPACKE_ssytrf_rook* and *LAPACKE_zsytrf_rook*. It also exists with a native Fortran interface as *dsytrf_rook*.

4.19.1129 LAPACKE_dsytrf_rook_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsytrf_rook_work(armpl_int_t matrix_layout, char uplo,
                                     armpl_int_t n, double *a,
                                     armpl_int_t lda, armpl_int_t *ipiv,
                                     double *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrf_rook_work*, *LAPACKE_dsytrf_rook_work*, *LAPACKE_ssytrf_rook_work* and *LAPACKE_zsytrf_rook_work*.

4.19.1130 LAPACKE_dsytrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsytrf_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, double *a, armpl_int_t lda,
                                armpl_int_t *ipiv, double *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrf_work*, *LAPACKE_dsytrf_work*, *LAPACKE_ssytrf_work* and *LAPACKE_zsytrf_work*.

4.19.1131 LAPACKE_dsytri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsytri(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, double *a, armpl_int_t lda,
                           const armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytri*, *LAPACKE_dsytri*, *LAPACKE_ssytri* and *LAPACKE_zsytri*. It also exists with a native Fortran interface as *dsytri*.

4.19.1132 LAPACKE_dsytri2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsytri2(armpl_int_t matrix_layout, char uplo,
                            armpl_int_t n, double *a, armpl_int_t lda,
                            const armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.1133 LAPACKE_dsytri2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsytri2_work(armpl_int_t matrix_layout, char uplo,
                                 armpl_int_t n, double *a, armpl_int_t lda,
                                 const armpl_int_t *ipiv, double *work,
                                 armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytri2_work*, *LAPACKE_dsytri2_work*, *LAPACKE_ssytri2_work* and *LAPACKE_zsytri2_work*.

4.19.1134 LAPACKE_dsytri2x

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsytri2x(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, double *a, armpl_int_t lda,
                             const armpl_int_t *ipiv, armpl_int_t nb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytri2x*, *LAPACKE_dsytri2x*, *LAPACKE_ssytri2x* and *LAPACKE_zsytri2x*. It also exists with a native Fortran interface as *dsytri2x*.

4.19.1135 LAPACKE_dsytri2x_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsytri2x_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, double *a, armpl_int_t lda,
                                  const armpl_int_t *ipiv, double *work,
                                  armpl_int_t nb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytri2x_work*, *LAPACKE_dsytri2x_work*, *LAPACKE_ssytri2x_work* and *LAPACKE_zsytri2x_work*.

4.19.1136 LAPACKE_dsytri_3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsytri_3(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, double *a, armpl_int_t lda,
                             const double *e, const armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytri_3*, *LAPACKE_dsytri_3*, *LAPACKE_ssytri_3* and *LAPACKE_zsytri_3*. It also exists with a native Fortran interface as *dsytri_3*.

4.19.1137 LAPACKE_dsytri_3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsytri_3_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, double *a, armpl_int_t lda,
                                  const double *e, const armpl_int_t *ipiv,
                                  double *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytri_3_work*, *LAPACKE_dsytri_3_work*, *LAPACKE_ssytri_3_work* and *LAPACKE_zsytri_3_work*.

4.19.1138 LAPACKE_dsytri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsytri_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, double *a, armpl_int_t lda,
                                const armpl_int_t *ipiv, double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytri_work*, *LAPACKE_dsytri_work*, *LAPACKE_ssytri_work* and *LAPACKE_zsytri_work*.

4.19.1139 LAPACKE_dsytrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsytrs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs, const double *a,
                           armpl_int_t lda, const armpl_int_t *ipiv,
                           double *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs*, *LAPACKE_dsytrs*, *LAPACKE_ssytrs* and *LAPACKE_zsytrs*. It also exists with a native Fortran interface as *dsytrs*.

4.19.1140 LAPACKE_dsytrs2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsytrs2(armpl_int_t matrix_layout, char uplo,
                            armpl_int_t n, armpl_int_t nrhs, const double *a,
                            armpl_int_t lda, const armpl_int_t *ipiv,
                            double *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs2*, *LAPACKE_dsytrs2*, *LAPACKE_ssytrs2* and *LAPACKE_zsytrs2*. It also exists with a native Fortran interface as *dsytrs2*.

4.19.1141 LAPACKE_dsytrs2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsytrs2_work(armpl_int_t matrix_layout, char uplo,
                                 armpl_int_t n, armpl_int_t nrhs,
                                 const double *a, armpl_int_t lda,
                                 const armpl_int_t *ipiv, double *b,
                                 armpl_int_t ldb, double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs2_work*, *LAPACKE_dsytrs2_work*, *LAPACKE_ssytrs2_work* and *LAPACKE_zsytrs2_work*.

4.19.1142 LAPACKE_dsytrs_3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsytrs_3(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, armpl_int_t nrhs, const double *a,
                             armpl_int_t lda, const double *e,
                             const armpl_int_t *ipiv, double *b,
                             armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs_3*, *LAPACKE_dsytrs_3*, *LAPACKE_ssytrs_3* and *LAPACKE_zsytrs_3*. It also exists with a native Fortran interface as *dsytrs_3*.

4.19.1143 LAPACKE_dsytrs_3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsytrs_3_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, armpl_int_t nrhs,
                                  const double *a, armpl_int_t lda,
                                  const double *e, const armpl_int_t *ipiv,
                                  double *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs_3_work*, *LAPACKE_dsytrs_3_work*, *LAPACKE_ssytrs_3_work* and *LAPACKE_zsytrs_3_work*.

4.19.1144 LAPACKE_dsytrs_aa

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsytrs_aa(armpl_int_t matrix_layout, char uplo,
                              armpl_int_t n, armpl_int_t nrhs,
                              const double *a, armpl_int_t lda,
                              const armpl_int_t *ipiv, double *b,
                              armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs_aa*, *LAPACKE_dsytrs_aa*, *LAPACKE_ssytrs_aa* and *LAPACKE_zsytrs_aa*. It also exists with a native Fortran interface as *dsytrs_aa*.

4.19.1145 LAPACKE_dsytrs_aa_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsytrs_aa_2stage(armpl_int_t matrix_layout, char uplo,
                                     armpl_int_t n, armpl_int_t nrhs,
                                     double *a, armpl_int_t lda, double *tb,
                                     armpl_int_t ltb, armpl_int_t *ipiv,
                                     armpl_int_t *ipiv2, double *b,
                                     armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs_aa_2stage*, *LAPACKE_dsytrs_aa_2stage*, *LAPACKE_ssytrs_aa_2stage* and *LAPACKE_zsytrs_aa_2stage*. It also exists with a native Fortran interface as *dsytrs_aa_2stage*.

4.19.1146 LAPACKE_dsytrs_aa_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsytrs_aa_work(armpl_int_t matrix_layout, char uplo,
                                   armpl_int_t n, armpl_int_t nrhs,
                                   const double *a, armpl_int_t lda,
                                   const armpl_int_t *ipiv, double *b,
                                   armpl_int_t ldb, double *work,
                                   armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs_aa_work*, *LAPACKE_dsytrs_aa_work*, *LAPACKE_ssytrs_aa_work* and *LAPACKE_zsytrs_aa_work*.

4.19.1147 LAPACKE_dsytrs_rook

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsytrs_rook(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const double *a, armpl_int_t lda,
                                const armpl_int_t *ipiv, double *b,
                                armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs_rook*, *LAPACKE_dsytrs_rook*, *LAPACKE_ssytrs_rook* and *LAPACKE_zsytrs_rook*. It also exists with a native Fortran interface as *dsytrs_rook*.

4.19.1148 LAPACKE_dsytrs_rook_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsytrs_rook_work(armpl_int_t matrix_layout, char uplo,
                                     armpl_int_t n, armpl_int_t nrhs,
                                     const double *a, armpl_int_t lda,
                                     const armpl_int_t *ipiv, double *b,
                                     armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs_rook_work*, *LAPACKE_dsytrs_rook_work*, *LAPACKE_ssytrs_rook_work* and *LAPACKE_zsytrs_rook_work*.

4.19.1149 LAPACKE_dsytrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dsytrs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const double *a, armpl_int_t lda,
                                const armpl_int_t *ipiv, double *b,
                                armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs_work*, *LAPACKE_dsytrs_work*, *LAPACKE_ssytrs_work* and *LAPACKE_zsytrs_work*.

4.19.1150 LAPACKE_dtbcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtbcon(armpl_int_t matrix_layout, char norm, char uplo,
                           char diag, armpl_int_t n, armpl_int_t kd,
                           const double *ab, armpl_int_t ldab,
                           double *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctbcon*, *LAPACKE_dtbcon*, *LAPACKE_stbcon* and *LAPACKE_ztbcon*. It also exists with a native Fortran interface as *dtbcon*.

4.19.1151 LAPACKE_dtbcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtbcon_work(armpl_int_t matrix_layout, char norm,
                                char uplo, char diag, armpl_int_t n,
                                armpl_int_t kd, const double *ab,
                                armpl_int_t ldab, double *rcond, double *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctbcon_work*, *LAPACKE_dtbcon_work*, *LAPACKE_stbcon_work* and *LAPACKE_ztbcon_work*.

4.19.1152 LAPACKE_dtbrfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtbrfs(armpl_int_t matrix_layout, char uplo, char trans,
                           char diag, armpl_int_t n, armpl_int_t kd,
                           armpl_int_t nrhs, const double *ab,
                           armpl_int_t ldab, const double *b, armpl_int_t ldb,
                           const double *x, armpl_int_t ldx, double *ferr,
                           double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctbrfs*, *LAPACKE_dtbrfs*, *LAPACKE_stbrfs* and *LAPACKE_ztbrfs*. It also exists with a native Fortran interface as *dtbrfs*.

4.19.1153 LAPACKE_dtbrfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtbrfs_work(armpl_int_t matrix_layout, char uplo,
                                char trans, char diag, armpl_int_t n,
                                armpl_int_t kd, armpl_int_t nrhs,
                                const double *ab, armpl_int_t ldab,
                                const double *b, armpl_int_t ldb,
                                const double *x, armpl_int_t ldx,
                                double *ferr, double *berr, double *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctbrfs_work*, *LAPACKE_dtbrfs_work*, *LAPACKE_stbrfs_work* and *LAPACKE_ztbrfs_work*.

4.19.1154 LAPACKE_dtbtrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtbtrs(armpl_int_t matrix_layout, char uplo, char trans,
                           char diag, armpl_int_t n, armpl_int_t kd,
                           armpl_int_t nrhs, const double *ab,
                           armpl_int_t ldab, double *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctbtrs*, *LAPACKE_dtbtrs*, *LAPACKE_stbtrs* and *LAPACKE_ztbtrs*. It also exists with a native Fortran interface as *dtbtrs*.

4.19.1155 LAPACKE_dtbtrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtbtrs_work(armpl_int_t matrix_layout, char uplo,
                                char trans, char diag, armpl_int_t n,
                                armpl_int_t kd, armpl_int_t nrhs,
                                const double *ab, armpl_int_t ldab, double *b,
                                armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctbtrs_work*, *LAPACKE_dtbtrs_work*, *LAPACKE_stbtrs_work* and *LAPACKE_ztbtrs_work*.

4.19.1156 LAPACKE_dtfsm

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtfsm(armpl_int_t matrix_layout, char transr, char side,
                          char uplo, char trans, char diag, armpl_int_t m,
                          armpl_int_t n, double alpha, const double *a,
                          double *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctfsm*, *LAPACKE_dtfsm*, *LAPACKE_stfsm* and *LAPACKE_ztfsm*. It also exists with a native Fortran interface as *dtfsm*.

4.19.1157 LAPACKE_dtfsm_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtfsm_work(armpl_int_t matrix_layout, char transr,
                               char side, char uplo, char trans, char diag,
                               armpl_int_t m, armpl_int_t n, double alpha,
                               const double *a, double *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctfsm_work*, *LAPACKE_dtfsm_work*, *LAPACKE_stfsm_work* and *LAPACKE_ztfsm_work*.

4.19.1158 LAPACKE_dtftri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtftri(armpl_int_t matrix_layout, char transr, char uplo,
                           char diag, armpl_int_t n, double *a);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctftri*, *LAPACKE_dtftri*, *LAPACKE_stftri* and *LAPACKE_ztftri*. It also exists with a native Fortran interface as *dtftri*.

4.19.1159 LAPACKE_dtftri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtftri_work(armpl_int_t matrix_layout, char transr,
                                char uplo, char diag, armpl_int_t n,
                                double *a);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctftri_work*, *LAPACKE_dtftri_work*, *LAPACKE_stftri_work* and *LAPACKE_ztftri_work*.

4.19.1160 LAPACKE_dtfttp

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtfttp(armpl_int_t matrix_layout, char transr, char uplo,
                           armpl_int_t n, const double *arf, double *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctfttp*, *LAPACKE_dtfttp*, *LAPACKE_stfttp* and *LAPACKE_ztfttp*. It also exists with a native Fortran interface as *dtfttp*.

4.19.1161 LAPACKE_dtfttp_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtfttp_work(armpl_int_t matrix_layout, char transr,
                                char uplo, armpl_int_t n, const double *arf,
                                double *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctfttp_work*, *LAPACKE_dtfttp_work*, *LAPACKE_stfttp_work* and *LAPACKE_ztfttp_work*.

4.19.1162 LAPACKE_dtfttr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtfttr(armpl_int_t matrix_layout, char transr, char uplo,
                           armpl_int_t n, const double *arf, double *a,
                           armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctfttr*, *LAPACKE_dtfttr*, *LAPACKE_stfttr* and *LAPACKE_ztfttr*. It also exists with a native Fortran interface as *dtfttr*.

4.19.1163 LAPACKE_dtfttr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtfttr_work(armpl_int_t matrix_layout, char transr,
                                char uplo, armpl_int_t n, const double *arf,
                                double *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctfttr_work*, *LAPACKE_dtfttr_work*, *LAPACKE_stfttr_work* and *LAPACKE_ztfttr_work*.

4.19.1164 LAPACKE_dtgevc

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtgevc(armpl_int_t matrix_layout, char side, char howmny,
                           const armpl_int_t *select, armpl_int_t n,
                           const double *s, armpl_int_t lds, const double *p,
                           armpl_int_t ldp, double *vl, armpl_int_t ldvl,
                           double *vr, armpl_int_t ldvr, armpl_int_t mm,
                           armpl_int_t *m);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgevc*, *LAPACKE_dtgevc*, *LAPACKE_stgevc* and *LAPACKE_ztgevc*. It also exists with a native Fortran interface as *dtgevc*.

4.19.1165 LAPACKE_dtgevc_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtgevc_work(armpl_int_t matrix_layout, char side,
                                char howmny, const armpl_int_t *select,
                                armpl_int_t n, const double *s,
                                armpl_int_t lds, const double *p,
                                armpl_int_t ldp, double *vl, armpl_int_t ldvl,
                                double *vr, armpl_int_t ldvr, armpl_int_t mm,
                                armpl_int_t *m, double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgevc_work*, *LAPACKE_dtgevc_work*, *LAPACKE_stgevc_work* and *LAPACKE_ztgevc_work*.

4.19.1166 LAPACKE_dtgexc

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtgexc(armpl_int_t matrix_layout, armpl_int_t wantq,
                           armpl_int_t wantz, armpl_int_t n, double *a,
```

(continues on next page)

(continued from previous page)

```
                                armpl_int_t lda, double *b, armpl_int_t ldb,
                                double *q, armpl_int_t ldq, double *z,
                                armpl_int_t ldz, armpl_int_t *ifst,
                                armpl_int_t *ilst);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgexc*, *LAPACKE_dtgexc*, *LAPACKE_stgexc* and *LAPACKE_ztgexc*. It also exists with a native Fortran interface as *dtgexc*.

4.19.1167 LAPACKE_dtgexc_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtgexc_work(armpl_int_t matrix_layout, armpl_int_t wantq,
                                armpl_int_t wantz, armpl_int_t n, double *a,
                                armpl_int_t lda, double *b, armpl_int_t ldb,
                                double *q, armpl_int_t ldq, double *z,
                                armpl_int_t ldz, armpl_int_t *ifst,
                                armpl_int_t *ilst, double *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgexc_work*, *LAPACKE_dtgexc_work*, *LAPACKE_stgexc_work* and *LAPACKE_ztgexc_work*.

4.19.1168 LAPACKE_dtgsen

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtgsen(armpl_int_t matrix_layout, armpl_int_t ijob,
                           armpl_int_t wantq, armpl_int_t wantz,
                           const armpl_int_t *select, armpl_int_t n,
                           double *a, armpl_int_t lda, double *b,
                           armpl_int_t ldb, double *alphar, double *alphai,
                           double *beta, double *q, armpl_int_t ldq,
                           double *z, armpl_int_t ldz, armpl_int_t *m,
                           double *pl, double *pr, double *dif);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgsen*, *LAPACKE_dtgsen*, *LAPACKE_stgsen* and *LAPACKE_ztgsen*. It also exists with a native Fortran interface as *dtgsen*.

4.19.1169 LAPACKE_dtgsen_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtgsen_work(armpl_int_t matrix_layout, armpl_int_t ijob,
                                armpl_int_t wantq, armpl_int_t wantz,
                                const armpl_int_t *select, armpl_int_t n,
                                double *a, armpl_int_t lda, double *b,
                                armpl_int_t ldb, double *alphar,
                                double *alphai, double *beta, double *q,
                                armpl_int_t ldq, double *z, armpl_int_t ldz,
                                armpl_int_t *m, double *pl, double *pr,
                                double *dif, double *work, armpl_int_t lwork,
                                armpl_int_t *iwork, armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgsen_work*, *LAPACKE_dtgsen_work*, *LAPACKE_stgsen_work* and *LAPACKE_ztgsen_work*.

4.19.1170 LAPACKE_dtgsja

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtgsja(armpl_int_t matrix_layout, char jobu, char jobv,
                           char jobq, armpl_int_t m, armpl_int_t p,
                           armpl_int_t n, armpl_int_t k, armpl_int_t l,
                           double *a, armpl_int_t lda, double *b,
                           armpl_int_t ldb, double tola, double tolb,
                           double *alpha, double *beta, double *u,
                           armpl_int_t ldu, double *v, armpl_int_t ldv,
                           double *q, armpl_int_t ldq, armpl_int_t *ncycle);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgsja*, *LAPACKE_dtgsja*, *LAPACKE_stgsja* and *LAPACKE_ztgsja*. It also exists with a native Fortran interface as *dtgsja*.

4.19.1171 LAPACKE_dtgsja_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtgsja_work(armpl_int_t matrix_layout, char jobu,
                                char jobv, char jobq, armpl_int_t m,
                                armpl_int_t p, armpl_int_t n, armpl_int_t k,
                                armpl_int_t l, double *a, armpl_int_t lda,
                                double *b, armpl_int_t ldb, double tola,
                                double tolb, double *alpha, double *beta,
                                double *u, armpl_int_t ldu, double *v,
                                armpl_int_t ldv, double *q, armpl_int_t ldq,
                                double *work, armpl_int_t *ncycle);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgsja_work*, *LAPACKE_dtgsja_work*, *LAPACKE_stgsja_work* and *LAPACKE_ztgsja_work*.

4.19.1172 LAPACKE_dtgsna

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtgsna(armpl_int_t matrix_layout, char job, char howmny,
                           const armpl_int_t *select, armpl_int_t n,
                           const double *a, armpl_int_t lda, const double *b,
                           armpl_int_t ldb, const double *vl,
                           armpl_int_t ldvl, const double *vr,
                           armpl_int_t ldvr, double *s, double *dif,
                           armpl_int_t mm, armpl_int_t *m);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgsna*, *LAPACKE_dtgsna*, *LAPACKE_stgsna* and *LAPACKE_ztgsna*. It also exists with a native Fortran interface as *dtgsna*.

4.19.1173 LAPACKE_dtgsna_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtgsna_work(armpl_int_t matrix_layout, char job,
                                char howmny, const armpl_int_t *select,
```

(continues on next page)

(continued from previous page)

```
                                armpl_int_t n, const double *a,
                                armpl_int_t lda, const double *b,
                                armpl_int_t ldb, const double *vl,
                                armpl_int_t ldvl, const double *vr,
                                armpl_int_t ldvr, double *s, double *dif,
                                armpl_int_t mm, armpl_int_t *m, double *work,
                                armpl_int_t lwork, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgsna_work*, *LAPACKE_dtgsna_work*, *LAPACKE_stgsna_work* and *LAPACKE_ztgsna_work*.

4.19.1174 LAPACKE_dtgsyl

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtgsyl(armpl_int_t matrix_layout, char trans,
                           armpl_int_t ijob, armpl_int_t m, armpl_int_t n,
                           const double *a, armpl_int_t lda, const double *b,
                           armpl_int_t ldb, double *c, armpl_int_t ldc,
                           const double *d, armpl_int_t ldd, const double *e,
                           armpl_int_t lde, double *f, armpl_int_t ldf,
                           double *scale, double *dif);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgsyl*, *LAPACKE_dtgsyl*, *LAPACKE_stgsyl* and *LAPACKE_ztgsyl*. It also exists with a native Fortran interface as *dtgsyl*.

4.19.1175 LAPACKE_dtgsyl_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtgsyl_work(armpl_int_t matrix_layout, char trans,
                                armpl_int_t ijob, armpl_int_t m,
                                armpl_int_t n, const double *a,
                                armpl_int_t lda, const double *b,
                                armpl_int_t ldb, double *c, armpl_int_t ldc,
                                const double *d, armpl_int_t ldd,
                                const double *e, armpl_int_t lde, double *f,
                                armpl_int_t ldf, double *scale, double *dif,
                                double *work, armpl_int_t lwork,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgsyl_work*, *LAPACKE_dtgsyl_work*, *LAPACKE_stgsyl_work* and *LAPACKE_ztgsyl_work*.

4.19.1176 LAPACKE_dtpcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtpcon(armpl_int_t matrix_layout, char norm, char uplo,
                           char diag, armpl_int_t n, const double *ap,
                           double *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpcon*, *LAPACKE_dtpcon*, *LAPACKE_stpcon* and *LAPACKE_ztpcon*. It also exists with a native Fortran interface as *dtpcon*.

4.19.1177 LAPACKE_dtpcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtpcon_work(armpl_int_t matrix_layout, char norm,
                                char uplo, char diag, armpl_int_t n,
                                const double *ap, double *rcond, double *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpcon_work*, *LAPACKE_dtpcon_work*, *LAPACKE_stpcon_work* and *LAPACKE_ztpcon_work*.

4.19.1178 LAPACKE_dtpmqrt

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtpmqrt(armpl_int_t matrix_layout, char side, char trans,
                            armpl_int_t m, armpl_int_t n, armpl_int_t k,
                            armpl_int_t l, armpl_int_t nb, const double *v,
                            armpl_int_t ldv, const double *t, armpl_int_t ldt,
                            double *a, armpl_int_t lda, double *b,
                            armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpmqrt*, *LAPACKE_dtpmqrt*, *LAPACKE_stpmqrt* and *LAPACKE_ztpmqrt*. It also exists with a native Fortran interface as *dtpmqrt*.

4.19.1179 LAPACKE_dtpmqrt_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtpmqrt_work(armpl_int_t matrix_layout, char side,
                                 char trans, armpl_int_t m, armpl_int_t n,
                                 armpl_int_t k, armpl_int_t l, armpl_int_t nb,
                                 const double *v, armpl_int_t ldv,
                                 const double *t, armpl_int_t ldt, double *a,
                                 armpl_int_t lda, double *b, armpl_int_t ldb,
                                 double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpmqrt_work*, *LAPACKE_dtpmqrt_work*, *LAPACKE_stpmqrt_work* and *LAPACKE_ztpmqrt_work*.

4.19.1180 LAPACKE_dtpqrt

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtpqrt(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t l, armpl_int_t nb,
                           double *a, armpl_int_t lda, double *b,
                           armpl_int_t ldb, double *t, armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpqrt*, *LAPACKE_dtpqrt*, *LAPACKE_stpqrt* and *LAPACKE_ztpqrt*. It also exists with a native Fortran interface as *dtpqrt*.

4.19.1181 LAPACKE_dtpqrt2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtpqrt2(armpl_int_t matrix_layout, armpl_int_t m,
                            armpl_int_t n, armpl_int_t l, double *a,
                            armpl_int_t lda, double *b, armpl_int_t ldb,
                            double *t, armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpqrt2*, *LAPACKE_dtpqrt2*, *LAPACKE_stpqrt2* and *LAPACKE_ztpqrt2*. It also exists with a native Fortran interface as *dtpqrt2*.

4.19.1182 LAPACKE_dtpqrt2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtpqrt2_work(armpl_int_t matrix_layout, armpl_int_t m,
                                 armpl_int_t n, armpl_int_t l, double *a,
                                 armpl_int_t lda, double *b, armpl_int_t ldb,
                                 double *t, armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpqrt2_work*, *LAPACKE_dtpqrt2_work*, *LAPACKE_stpqrt2_work* and *LAPACKE_ztpqrt2_work*.

4.19.1183 LAPACKE_dtpqrt_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtpqrt_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t l, armpl_int_t nb,
                                double *a, armpl_int_t lda, double *b,
                                armpl_int_t ldb, double *t, armpl_int_t ldt,
                                double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpqrt_work*, *LAPACKE_dtpqrt_work*, *LAPACKE_stpqrt_work* and *LAPACKE_ztpqrt_work*.

4.19.1184 LAPACKE_dtprfb

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtprfb(armpl_int_t matrix_layout, char side, char trans,
                           char direct, char storev, armpl_int_t m,
                           armpl_int_t n, armpl_int_t k, armpl_int_t l,
                           const double *v, armpl_int_t ldv, const double *t,
                           armpl_int_t ldt, double *a, armpl_int_t lda,
                           double *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctprfb*, *LAPACKE_dtprfb*, *LAPACKE_stprfb* and *LAPACKE_ztprfb*. It also exists with a native Fortran interface as *dtprfb*.

4.19.1185 LAPACKE_dtprfb_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtprfb_work(armpl_int_t matrix_layout, char side,
                                char trans, char direct, char storev,
                                armpl_int_t m, armpl_int_t n, armpl_int_t k,
                                armpl_int_t l, const double *v,
                                armpl_int_t ldv, const double *t,
                                armpl_int_t ldt, double *a, armpl_int_t lda,
                                double *b, armpl_int_t ldb, double *work,
                                armpl_int_t ldwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctprfb_work*, *LAPACKE_dtprfb_work*, *LAPACKE_stprfb_work* and *LAPACKE_ztprfb_work*.

4.19.1186 LAPACKE_dtprfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtprfs(armpl_int_t matrix_layout, char uplo, char trans,
                           char diag, armpl_int_t n, armpl_int_t nrhs,
                           const double *ap, const double *b, armpl_int_t ldb,
                           const double *x, armpl_int_t ldx, double *ferr,
                           double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctprfs*, *LAPACKE_dtprfs*, *LAPACKE_stprfs* and *LAPACKE_ztprfs*. It also exists with a native Fortran interface as *dtprfs*.

4.19.1187 LAPACKE_dtprfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtprfs_work(armpl_int_t matrix_layout, char uplo,
                                char trans, char diag, armpl_int_t n,
                                armpl_int_t nrhs, const double *ap,
                                const double *b, armpl_int_t ldb,
                                const double *x, armpl_int_t ldx,
                                double *ferr, double *berr, double *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctprfs_work*, *LAPACKE_dtprfs_work*, *LAPACKE_stprfs_work* and *LAPACKE_ztprfs_work*.

4.19.1188 LAPACKE_dtptri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtptri(armpl_int_t matrix_layout, char uplo, char diag,
                           armpl_int_t n, double *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctptri*, *LAPACKE_dtptri*, *LAPACKE_stptri* and *LAPACKE_ztptri*. It also exists with a native Fortran interface as *dtptri*.

4.19.1189 LAPACKE_dtptri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtptri_work(armpl_int_t matrix_layout, char uplo,
                                char diag, armpl_int_t n, double *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctptri_work*, *LAPACKE_dtptri_work*, *LAPACKE_stptri_work* and *LAPACKE_ztptri_work*.

4.19.1190 LAPACKE_dtptrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtptrs(armpl_int_t matrix_layout, char uplo, char trans,
                           char diag, armpl_int_t n, armpl_int_t nrhs,
                           const double *ap, double *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctptrs*, *LAPACKE_dtptrs*, *LAPACKE_stptrs* and *LAPACKE_ztptrs*. It also exists with a native Fortran interface as *dtptrs*.

4.19.1191 LAPACKE_dtptrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtptrs_work(armpl_int_t matrix_layout, char uplo,
                                char trans, char diag, armpl_int_t n,
                                armpl_int_t nrhs, const double *ap, double *b,
                                armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctptrs_work*, *LAPACKE_dtptrs_work*, *LAPACKE_stptrs_work* and *LAPACKE_ztptrs_work*.

4.19.1192 LAPACKE_dtpttf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtpttf(armpl_int_t matrix_layout, char transr, char uplo,
                           armpl_int_t n, const double *ap, double *arf);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpttf*, *LAPACKE_dtpttf*, *LAPACKE_stpttf* and *LAPACKE_ztpttf*. It also exists with a native Fortran interface as *dtpttf*.

4.19.1193 LAPACKE_dtpttf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtpttf_work(armpl_int_t matrix_layout, char transr,
                                char uplo, armpl_int_t n, const double *ap,
                                double *arf);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.1194 LAPACKE_dtpttr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtpttr(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, const double *ap, double *a,
                           armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpttr*, *LAPACKE_dtpttr*, *LAPACKE_stpttr* and *LAPACKE_ztpttr*. It also exists with a native Fortran interface as *dtpttr*.

4.19.1195 LAPACKE_dtpttr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtpttr_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, const double *ap, double *a,
                                armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpttr_work*, *LAPACKE_dtpttr_work*, *LAPACKE_stpttr_work* and *LAPACKE_ztpttr_work*.

4.19.1196 LAPACKE_dtrcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtrcon(armpl_int_t matrix_layout, char norm, char uplo,
                           char diag, armpl_int_t n, const double *a,
                           armpl_int_t lda, double *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrcon*, *LAPACKE_dtrcon*, *LAPACKE_strcon* and *LAPACKE_ztrcon*. It also exists with a native Fortran interface as *dtrcon*.

4.19.1197 LAPACKE_dtrcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtrcon_work(armpl_int_t matrix_layout, char norm,
                                char uplo, char diag, armpl_int_t n,
                                const double *a, armpl_int_t lda,
                                double *rcond, double *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrcon_work*, *LAPACKE_dtrcon_work*, *LAPACKE_strcon_work* and *LAPACKE_ztrcon_work*.

4.19.1198 LAPACKE_dtrevc

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtrevc(armpl_int_t matrix_layout, char side, char howmny,
                           armpl_int_t *select, armpl_int_t n,
                           const double *t, armpl_int_t ldt, double *vl,
                           armpl_int_t ldvl, double *vr, armpl_int_t ldvr,
                           armpl_int_t mm, armpl_int_t *m);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrevc*, *LAPACKE_dtrevc*, *LAPACKE_strevc* and *LAPACKE_ztrevc*. It also exists with a native Fortran interface as *dtrevc*.

4.19.1199 LAPACKE_dtrevc_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtrevc_work(armpl_int_t matrix_layout, char side,
                                char howmny, armpl_int_t *select,
                                armpl_int_t n, const double *t,
                                armpl_int_t ldt, double *vl, armpl_int_t ldvl,
                                double *vr, armpl_int_t ldvr, armpl_int_t mm,
                                armpl_int_t *m, double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrevc_work*, *LAPACKE_dtrevc_work*, *LAPACKE_strevc_work* and *LAPACKE_ztrevc_work*.

4.19.1200 LAPACKE_dtrexc

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtrexc(armpl_int_t matrix_layout, char compq,
                           armpl_int_t n, double *t, armpl_int_t ldt,
                           double *q, armpl_int_t ldq, armpl_int_t *ifst,
                           armpl_int_t *ilst);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrexc*, *LAPACKE_dtrexc*, *LAPACKE_strexc* and *LAPACKE_ztrexc*. It also exists with a native Fortran interface as *dtrexc*.

4.19.1201 LAPACKE_dtrexc_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtrexc_work(armpl_int_t matrix_layout, char compq,
                                armpl_int_t n, double *t, armpl_int_t ldt,
                                double *q, armpl_int_t ldq, armpl_int_t *ifst,
                                armpl_int_t *ilst, double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrexc_work*, *LAPACKE_dtrexc_work*, *LAPACKE_strexc_work* and *LAPACKE_ztrexc_work*.

4.19.1202 LAPACKE_dtrrfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtrrfs(armpl_int_t matrix_layout, char uplo, char trans,
                           char diag, armpl_int_t n, armpl_int_t nrhs,
                           const double *a, armpl_int_t lda, const double *b,
                           armpl_int_t ldb, const double *x, armpl_int_t ldx,
                           double *ferr, double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrrfs*, *LAPACKE_dtrrfs*, *LAPACKE_strrfs* and *LAPACKE_ztrrfs*. It also exists with a native Fortran interface as *dtrrfs*.

4.19.1203 LAPACKE_dtrrfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtrrfs_work(armpl_int_t matrix_layout, char uplo,
                                char trans, char diag, armpl_int_t n,
                                armpl_int_t nrhs, const double *a,
                                armpl_int_t lda, const double *b,
                                armpl_int_t ldb, const double *x,
                                armpl_int_t ldx, double *ferr, double *berr,
                                double *work, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrrfs_work*, *LAPACKE_dtrrfs_work*, *LAPACKE_strrfs_work* and *LAPACKE_ztrrfs_work*.

4.19.1204 LAPACKE_dtrsen

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtrsen(armpl_int_t matrix_layout, char job, char compq,
                           const armpl_int_t *select, armpl_int_t n,
                           double *t, armpl_int_t ldt, double *q,
                           armpl_int_t ldq, double *wr, double *wi,
                           armpl_int_t *m, double *s, double *sep);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrsen*, *LAPACKE_dtrsen*, *LAPACKE_strsen* and *LAPACKE_ztrsen*. It also exists with a native Fortran interface as *dtrsen*.

4.19.1205 LAPACKE_dtrsen_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtrsen_work(armpl_int_t matrix_layout, char job,
                                char compq, const armpl_int_t *select,
                                armpl_int_t n, double *t, armpl_int_t ldt,
                                double *q, armpl_int_t ldq, double *wr,
                                double *wi, armpl_int_t *m, double *s,
                                double *sep, double *work, armpl_int_t lwork,
                                armpl_int_t *iwork, armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrsen_work*, *LAPACKE_dtrsen_work*, *LAPACKE_strsen_work* and *LAPACKE_ztrsen_work*.

4.19.1206 LAPACKE_dtrsna

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtrsna(armpl_int_t matrix_layout, char job, char howmny,
                           const armpl_int_t *select, armpl_int_t n,
                           const double *t, armpl_int_t ldt, const double *vl,
                           armpl_int_t ldvl, const double *vr,
                           armpl_int_t ldvr, double *s, double *sep,
                           armpl_int_t mm, armpl_int_t *m);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrsna*, *LAPACKE_dtrsna*, *LAPACKE_strsna* and *LAPACKE_ztrsna*. It also exists with a native Fortran interface as *dtrsna*.

4.19.1207 LAPACKE_dtrsna_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtrsna_work(armpl_int_t matrix_layout, char job,
                                char howmny, const armpl_int_t *select,
                                armpl_int_t n, const double *t,
```

(continues on next page)

(continued from previous page)

```
                                armpl_int_t ldt, const double *vl,
                                armpl_int_t ldvl, const double *vr,
                                armpl_int_t ldvr, double *s, double *sep,
                                armpl_int_t mm, armpl_int_t *m, double *work,
                                armpl_int_t ldwork, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrsna_work*, *LAPACKE_dtrsna_work*, *LAPACKE_strsna_work* and *LAPACKE_ztrsna_work*.

4.19.1208 LAPACKE_dtrsyl

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtrsyl(armpl_int_t matrix_layout, char trana, char tranb,
                           armpl_int_t isgn, armpl_int_t m, armpl_int_t n,
                           const double *a, armpl_int_t lda, const double *b,
                           armpl_int_t ldb, double *c, armpl_int_t ldc,
                           double *scale);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrsyl*, *LAPACKE_dtrsyl*, *LAPACKE_strsyl* and *LAPACKE_ztrsyl*. It also exists with a native Fortran interface as *dtrsyl*.

4.19.1209 LAPACKE_dtrsyl_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtrsyl_work(armpl_int_t matrix_layout, char trana,
                                char tranb, armpl_int_t isgn, armpl_int_t m,
                                armpl_int_t n, const double *a,
                                armpl_int_t lda, const double *b,
                                armpl_int_t ldb, double *c, armpl_int_t ldc,
                                double *scale);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrsyl_work*, *LAPACKE_dtrsyl_work*, *LAPACKE_strsyl_work* and *LAPACKE_ztrsyl_work*.

4.19.1210 LAPACKE_dtrtri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtrtri(armpl_int_t matrix_layout, char uplo, char diag,
                           armpl_int_t n, double *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrtri*, *LAPACKE_dtrtri*, *LAPACKE_strtri* and *LAPACKE_ztrtri*. It also exists with a native Fortran interface as *dtrtri*.

4.19.1211 LAPACKE_dtrtri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtrtri_work(armpl_int_t matrix_layout, char uplo,
                                char diag, armpl_int_t n, double *a,
                                armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrtri_work*, *LAPACKE_dtrtri_work*, *LAPACKE_strtri_work* and *LAPACKE_ztrtri_work*.

4.19.1212 LAPACKE_dtrtrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtrtrs(armpl_int_t matrix_layout, char uplo, char trans,
                           char diag, armpl_int_t n, armpl_int_t nrhs,
                           const double *a, armpl_int_t lda, double *b,
                           armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrtrs*, *LAPACKE_dtrtrs*, *LAPACKE_strtrs* and *LAPACKE_ztrtrs*. It also exists with a native Fortran interface as *dtrtrs*.

4.19.1213 LAPACKE_dtrtrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtrtrs_work(armpl_int_t matrix_layout, char uplo,
                                char trans, char diag, armpl_int_t n,
                                armpl_int_t nrhs, const double *a,
                                armpl_int_t lda, double *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrtrs_work*, *LAPACKE_dtrtrs_work*, *LAPACKE_strtrs_work* and *LAPACKE_ztrtrs_work*.

4.19.1214 LAPACKE_dtrttf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtrttf(armpl_int_t matrix_layout, char transr, char uplo,
                           armpl_int_t n, const double *a, armpl_int_t lda,
                           double *arf);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrttf*, *LAPACKE_dtrttf*, *LAPACKE_strttf* and *LAPACKE_ztrttf*. It also exists with a native Fortran interface as *dtrttf*.

4.19.1215 LAPACKE_dtrttf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtrttf_work(armpl_int_t matrix_layout, char transr,
                                char uplo, armpl_int_t n, const double *a,
                                armpl_int_t lda, double *arf);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrttf_work*, *LAPACKE_dtrttf_work*, *LAPACKE_strttf_work* and *LAPACKE_ztrttf_work*.

4.19.1216 LAPACKE_dtrttp

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtrttp(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, const double *a, armpl_int_t lda,
                           double *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrttp*, *LAPACKE_dtrttp*, *LAPACKE_strttp* and *LAPACKE_ztrttp*. It also exists with a native Fortran interface as *dtrttp*.

4.19.1217 LAPACKE_dtrttp_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtrttp_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, const double *a,
                                armpl_int_t lda, double *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrttp_work*, *LAPACKE_dtrttp_work*, *LAPACKE_strttp_work* and *LAPACKE_ztrttp_work*.

4.19.1218 LAPACKE_dtzrzf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtzrzf(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, double *a, armpl_int_t lda,
                           double *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctzrzf*, *LAPACKE_dtzrzf*, *LAPACKE_stzrzf* and *LAPACKE_ztzrzf*. It also exists with a native Fortran interface as *dtzrzf*.

4.19.1219 LAPACKE_dtzrzf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_dtzrzf_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, double *a, armpl_int_t lda,
                                double *tau, double *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctzrzf_work*, *LAPACKE_dtzrzf_work*, *LAPACKE_stzrzf_work* and *LAPACKE_ztzrzf_work*.

4.19.1220 LAPACKE_ilaver

Syntax

```
#include "armpl.h"

void LAPACKE_ilaver(armpl_int_t *vers_major, armpl_int_t *vers_minor,
                    armpl_int_t *vers_patch);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

4.19.1221 LAPACKE_sbbcsd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sbbcsd(armpl_int_t matrix_layout, char jobu1, char jobu2,
                           char jobv1t, char jobv2t, char trans,
                           armpl_int_t m, armpl_int_t p, armpl_int_t q,
                           float *theta, float *phi, float *u1,
                           armpl_int_t ldu1, float *u2, armpl_int_t ldu2,
                           float *v1t, armpl_int_t ldv1t, float *v2t,
                           armpl_int_t ldv2t, float *b11d, float *b11e,
                           float *b12d, float *b12e, float *b21d, float *b21e,
                           float *b22d, float *b22e);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cbbcsd*, *LAPACKE_dbbcsd*, *LAPACKE_sbbcsd* and *LAPACKE_zbbcsd*. It also exists with a native Fortran interface as *sbbcsd*.

4.19.1222 LAPACKE_sbbcsd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sbbcsd_work(armpl_int_t matrix_layout, char jobu1,
                                char jobu2, char jobv1t, char jobv2t,
                                char trans, armpl_int_t m, armpl_int_t p,
                                armpl_int_t q, float *theta, float *phi,
                                float *u1, armpl_int_t ldu1, float *u2,
                                armpl_int_t ldu2, float *v1t,
                                armpl_int_t ldv1t, float *v2t,
                                armpl_int_t ldv2t, float *b11d, float *b11e,
                                float *b12d, float *b12e, float *b21d,
                                float *b21e, float *b22d, float *b22e,
                                float *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cbbcsd_work*, *LAPACKE_dbbcsd_work*, *LAPACKE_sbbcsd_work* and *LAPACKE_zbbcsd_work*.

4.19.1223 LAPACKE_sbdsdc

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sbdsdc(armpl_int_t matrix_layout, char uplo, char compq,
                           armpl_int_t n, float *d, float *e, float *u,
                           armpl_int_t ldu, float *vt, armpl_int_t ldvt,
                           float *q, armpl_int_t *iq);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dbdsdc* and *LAPACKE_sbdsdc*. It also exists with a native Fortran interface as *sbdsdc*.

4.19.1224 LAPACKE_sbdsdc_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sbdsdc_work(armpl_int_t matrix_layout, char uplo,
                                char compq, armpl_int_t n, float *d, float *e,
                                float *u, armpl_int_t ldu, float *vt,
                                armpl_int_t ldvt, float *q, armpl_int_t *iq,
                                float *work, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dbdsdc_work* and *LAPACKE_sbdsdc_work*.

4.19.1225 LAPACKE_sbdsqr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sbdsqr(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t ncvt, armpl_int_t nru,
                           armpl_int_t ncc, float *d, float *e, float *vt,
                           armpl_int_t ldvt, float *u, armpl_int_t ldu,
                           float *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cbdsqr*, *LAPACKE_dbdsqr*, *LAPACKE_sbdsqr* and *LAPACKE_zbdsqr*. It also exists with a native Fortran interface as *sbdsqr*.

4.19.1226 LAPACKE_sbdsqr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sbdsqr_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t ncvt,
                                armpl_int_t nru, armpl_int_t ncc, float *d,
                                float *e, float *vt, armpl_int_t ldvt,
                                float *u, armpl_int_t ldu, float *c,
                                armpl_int_t ldc, float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cbdsqr_work*, *LAPACKE_dbdsqr_work*, *LAPACKE_sbdsqr_work* and *LAPACKE_zbdsqr_work*.

4.19.1227 LAPACKE_sbdsvdx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sbdsvdx(armpl_int_t matrix_layout, char uplo, char jobz,
                            char range, armpl_int_t n, float *d, float *e,
                            float vl, float vu, armpl_int_t il,
                            armpl_int_t iu, armpl_int_t *ns, float *s,
                            float *z, armpl_int_t ldz, armpl_int_t *superb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dbdsvdx* and *LAPACKE_sbdsvdx*. It also exists with a native Fortran interface as *sbdsvdx*.

4.19.1228 LAPACKE_sbdsvdx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sbdsvdx_work(armpl_int_t matrix_layout, char uplo,
                                 char jobz, char range, armpl_int_t n,
                                 float *d, float *e, float vl, float vu,
                                 armpl_int_t il, armpl_int_t iu,
                                 armpl_int_t *ns, float *s, float *z,
                                 armpl_int_t ldz, float *work,
                                 armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dbdsvdx_work* and *LAPACKE_sbdsvdx_work*.

4.19.1229 LAPACKE_sdisna

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sdisna(char job, armpl_int_t m, armpl_int_t n,
                           const float *d, float *sep);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ddisna* and *LAPACKE_sdisna*. It also exists with a native Fortran interface as *sdisna*.

4.19.1230 LAPACKE_sdisna_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sdisna_work(char job, armpl_int_t m, armpl_int_t n,
                                const float *d, float *sep);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ddisna_work* and *LAPACKE_sdisna_work*.

4.19.1231 LAPACKE_sgbbrd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgbbrd(armpl_int_t matrix_layout, char vect,
                           armpl_int_t m, armpl_int_t n, armpl_int_t ncc,
                           armpl_int_t kl, armpl_int_t ku, float *ab,
                           armpl_int_t ldab, float *d, float *e, float *q,
                           armpl_int_t ldq, float *pt, armpl_int_t ldpt,
                           float *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbbrd*, *LAPACKE_dgbbrd*, *LAPACKE_sgbbrd* and *LAPACKE_zgbbrd*. It also exists with a native Fortran interface as *sgbbrd*.

4.19.1232 LAPACKE_sgbbrd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgbbrd_work(armpl_int_t matrix_layout, char vect,
                                armpl_int_t m, armpl_int_t n, armpl_int_t ncc,
                                armpl_int_t kl, armpl_int_t ku, float *ab,
                                armpl_int_t ldab, float *d, float *e,
                                float *q, armpl_int_t ldq, float *pt,
                                armpl_int_t ldpt, float *c, armpl_int_t ldc,
                                float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbbrd_work*, *LAPACKE_dgbbrd_work*, *LAPACKE_sgbbrd_work* and *LAPACKE_zgbbrd_work*.

4.19.1233 LAPACKE_sgbcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgbcon(armpl_int_t matrix_layout, char norm,
                           armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                           const float *ab, armpl_int_t ldab,
                           const armpl_int_t *ipiv, float anorm,
                           float *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbcon*, *LAPACKE_dgbcon*, *LAPACKE_sgbcon* and *LAPACKE_zgbcon*. It also exists with a native Fortran interface as *sgbcon*.

4.19.1234 LAPACKE_sgbcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgbcon_work(armpl_int_t matrix_layout, char norm,
                                armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                                const float *ab, armpl_int_t ldab,
                                const armpl_int_t *ipiv, float anorm,
                                float *rcond, float *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbcon_work*, *LAPACKE_dgbcon_work*, *LAPACKE_sgbcon_work* and *LAPACKE_zgbcon_work*.

4.19.1235 LAPACKE_sgbequ

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgbequ(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                           const float *ab, armpl_int_t ldab, float *r,
                           float *c, float *rowcnd, float *colcnd,
                           float *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbequ*, *LAPACKE_dgbequ*, *LAPACKE_sgbequ* and *LAPACKE_zgbequ*. It also exists with a native Fortran interface as *sgbequ*.

4.19.1236 LAPACKE_sgbequ_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgbequ_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                                const float *ab, armpl_int_t ldab, float *r,
                                float *c, float *rowcnd, float *colcnd,
                                float *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbequ_work*, *LAPACKE_dgbequ_work*, *LAPACKE_sgbequ_work* and *LAPACKE_zgbequ_work*.

4.19.1237 LAPACKE_sgbequb

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgbequb(armpl_int_t matrix_layout, armpl_int_t m,
                            armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                            const float *ab, armpl_int_t ldab, float *r,
                            float *c, float *rowcnd, float *colcnd,
                            float *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbequb*, *LAPACKE_dgbequb*, *LAPACKE_sgbequb* and *LAPACKE_zgbequb*. It also exists with a native Fortran interface as *sgbequb*.

4.19.1238 LAPACKE_sgbequb_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgbequb_work(armpl_int_t matrix_layout, armpl_int_t m,
                                 armpl_int_t n, armpl_int_t kl,
                                 armpl_int_t ku, const float *ab,
                                 armpl_int_t ldab, float *r, float *c,
                                 float *rowcnd, float *colcnd, float *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbequb_work*, *LAPACKE_dgbequb_work*, *LAPACKE_sgbequb_work* and *LAPACKE_zgbequb_work*.

4.19.1239 LAPACKE_sgbrfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgbrfs(armpl_int_t matrix_layout, char trans,
                           armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                           armpl_int_t nrhs, const float *ab,
                           armpl_int_t ldab, const float *afb,
                           armpl_int_t ldafb, const armpl_int_t *ipiv,
                           const float *b, armpl_int_t ldb, float *x,
                           armpl_int_t ldx, float *ferr, float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbrfs*, *LAPACKE_dgbrfs*, *LAPACKE_sgbrfs* and *LAPACKE_zgbrfs*. It also exists with a native Fortran interface as *sgbrfs*.

4.19.1240 LAPACKE_sgbrfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgbrfs_work(armpl_int_t matrix_layout, char trans,
                                armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
```

(continues on next page)

(continued from previous page)

```
                                armpl_int_t nrhs, const float *ab,
                                armpl_int_t ldab, const float *afb,
                                armpl_int_t ldafb, const armpl_int_t *ipiv,
                                const float *b, armpl_int_t ldb, float *x,
                                armpl_int_t ldx, float *ferr, float *berr,
                                float *work, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbrfs_work*, *LAPACKE_dgbrfs_work*, *LAPACKE_sgbrfs_work* and *LAPACKE_zgbrfs_work*.

4.19.1241 LAPACKE_sgbrfsx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgbrfsx(armpl_int_t matrix_layout, char trans, char equed,
                            armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                            armpl_int_t nrhs, const float *ab,
                            armpl_int_t ldab, const float *afb,
                            armpl_int_t ldafb, const armpl_int_t *ipiv,
                            const float *r, const float *c, const float *b,
                            armpl_int_t ldb, float *x, armpl_int_t ldx,
                            float *rcond, float *berr, armpl_int_t n_err_bnds,
                            float *err_bnds_norm, float *err_bnds_comp,
                            armpl_int_t nparams, float *params);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbrfsx*, *LAPACKE_dgbrfsx*, *LAPACKE_sgbrfsx* and *LAPACKE_zgbrfsx*. It also exists with a native Fortran interface as *sgbrfsx*.

4.19.1242 LAPACKE_sgbrfsx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgbrfsx_work(armpl_int_t matrix_layout, char trans,
                                 char equed, armpl_int_t n, armpl_int_t kl,
                                 armpl_int_t ku, armpl_int_t nrhs,
                                 const float *ab, armpl_int_t ldab,
                                 const float *afb, armpl_int_t ldafb,
                                 const armpl_int_t *ipiv, const float *r,
                                 const float *c, const float *b,
                                 armpl_int_t ldb, float *x, armpl_int_t ldx,
                                 float *rcond, float *berr,
                                 armpl_int_t n_err_bnds, float *err_bnds_norm,
                                 float *err_bnds_comp, armpl_int_t nparams,
                                 float *params, float *work,
                                 armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbrfsx_work*, *LAPACKE_dgbrfsx_work*, *LAPACKE_sgbrfsx_work* and *LAPACKE_zgbrfsx_work*.

4.19.1243 LAPACKE_sgbsv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgbsv(armpl_int_t matrix_layout, armpl_int_t n,
                          armpl_int_t kl, armpl_int_t ku, armpl_int_t nrhs,
                          float *ab, armpl_int_t ldab, armpl_int_t *ipiv,
                          float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbsv*, *LAPACKE_dgbsv*, *LAPACKE_sgbsv* and *LAPACKE_zgbsv*. It also exists with a native Fortran interface as *sgbsv*.

4.19.1244 LAPACKE_sgbsv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgbsv_work(armpl_int_t matrix_layout, armpl_int_t n,
                               armpl_int_t kl, armpl_int_t ku,
                               armpl_int_t nrhs, float *ab, armpl_int_t ldab,
                               armpl_int_t *ipiv, float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbsv_work*, *LAPACKE_dgbsv_work*, *LAPACKE_sgbsv_work* and *LAPACKE_zgbsv_work*.

4.19.1245 LAPACKE_sgbsvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgbsvx(armpl_int_t matrix_layout, char fact, char trans,
                           armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                           armpl_int_t nrhs, float *ab, armpl_int_t ldab,
                           float *afb, armpl_int_t ldafb, armpl_int_t *ipiv,
                           char *equed, float *r, float *c, float *b,
                           armpl_int_t ldb, float *x, armpl_int_t ldx,
                           float *rcond, float *ferr, float *berr,
                           float *rpivot);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbsvx*, *LAPACKE_dgbsvx*, *LAPACKE_sgbsvx* and *LAPACKE_zgbsvx*. It also exists with a native Fortran interface as *sgbsvx*.

4.19.1246 LAPACKE_sgbsvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgbsvx_work(armpl_int_t matrix_layout, char fact,
                                char trans, armpl_int_t n, armpl_int_t kl,
                                armpl_int_t ku, armpl_int_t nrhs, float *ab,
                                armpl_int_t ldab, float *afb,
                                armpl_int_t ldafb, armpl_int_t *ipiv,
                                char *equed, float *r, float *c, float *b,
                                armpl_int_t ldb, float *x, armpl_int_t ldx,
                                float *rcond, float *ferr, float *berr,
                                float *work, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbsvx_work*, *LAPACKE_dgbsvx_work*, *LAPACKE_sgbsvx_work* and *LAPACKE_zgbsvx_work*.

4.19.1247 LAPACKE_sgbsvxx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgbsvxx(armpl_int_t matrix_layout, char fact, char trans,
                            armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                            armpl_int_t nrhs, float *ab, armpl_int_t ldab,
                            float *afb, armpl_int_t ldafb, armpl_int_t *ipiv,
                            char *equed, float *r, float *c, float *b,
                            armpl_int_t ldb, float *x, armpl_int_t ldx,
                            float *rcond, float *rpvgrw, float *berr,
                            armpl_int_t n_err_bnds, float *err_bnds_norm,
                            float *err_bnds_comp, armpl_int_t nparams,
                            float *params);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbsvxx*, *LAPACKE_dgbsvxx*, *LAPACKE_sgbsvxx* and *LAPACKE_zgbsvxx*. It also exists with a native Fortran interface as *sgbsvxx*.

4.19.1248 LAPACKE_sgbsvxx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgbsvxx_work(armpl_int_t matrix_layout, char fact,
                                 char trans, armpl_int_t n, armpl_int_t kl,
                                 armpl_int_t ku, armpl_int_t nrhs, float *ab,
                                 armpl_int_t ldab, float *afb,
                                 armpl_int_t ldafb, armpl_int_t *ipiv,
                                 char *equed, float *r, float *c, float *b,
                                 armpl_int_t ldb, float *x, armpl_int_t ldx,
                                 float *rcond, float *rpvgrw, float *berr,
                                 armpl_int_t n_err_bnds, float *err_bnds_norm,
                                 float *err_bnds_comp, armpl_int_t nparams,
                                 float *params, float *work,
                                 armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbsvxx_work*, *LAPACKE_dgbsvxx_work*, *LAPACKE_sgbsvxx_work* and *LAPACKE_zgbsvxx_work*.

4.19.1249 LAPACKE_sgbtrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgbtrf(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                           float *ab, armpl_int_t ldab, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbtrf*, *LAPACKE_dgbtrf*, *LAPACKE_sgbtrf* and *LAPACKE_zgbtrf*. It also exists with a native Fortran interface as *sgbtrf*.

4.19.1250 LAPACKE_sgbtrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgbtrf_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                                float *ab, armpl_int_t ldab,
                                armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbtrf_work*, *LAPACKE_dgbtrf_work*, *LAPACKE_sgbtrf_work* and *LAPACKE_zgbtrf_work*.

4.19.1251 LAPACKE_sgbtrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgbtrs(armpl_int_t matrix_layout, char trans,
                           armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                           armpl_int_t nrhs, const float *ab,
                           armpl_int_t ldab, const armpl_int_t *ipiv,
                           float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbtrs*, *LAPACKE_dgbtrs*, *LAPACKE_sgbtrs* and *LAPACKE_zgbtrs*. It also exists with a native Fortran interface as *sgbtrs*.

4.19.1252 LAPACKE_sgbtrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgbtrs_work(armpl_int_t matrix_layout, char trans,
                                armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                                armpl_int_t nrhs, const float *ab,
                                armpl_int_t ldab, const armpl_int_t *ipiv,
                                float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbtrs_work*, *LAPACKE_dgbtrs_work*, *LAPACKE_sgbtrs_work* and *LAPACKE_zgbtrs_work*.

4.19.1253 LAPACKE_sgebak

4.19.1254 LAPACKE_sgebak_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgebak_work(armpl_int_t matrix_layout, char job,
                                char side, armpl_int_t n, armpl_int_t ilo,
                                armpl_int_t ihi, const float *scale,
                                armpl_int_t m, float *v, armpl_int_t ldv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgebak_work*, *LAPACKE_dgebak_work*, *LAPACKE_sgebak_work* and *LAPACKE_zgebak_work*.

4.19.1255 LAPACKE_sgebal

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgebal(armpl_int_t matrix_layout, char job, armpl_int_t n,
                           float *a, armpl_int_t lda, armpl_int_t *ilo,
                           armpl_int_t *ihi, float *scale);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgebal*, *LAPACKE_dgebal*, *LAPACKE_sgebal* and *LAPACKE_zgebal*. It also exists with a native Fortran interface as *sgebal*.

4.19.1256 LAPACKE_sgebal_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgebal_work(armpl_int_t matrix_layout, char job,
                                armpl_int_t n, float *a, armpl_int_t lda,
                                armpl_int_t *ilo, armpl_int_t *ihi,
                                float *scale);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgebal_work*, *LAPACKE_dgebal_work*, *LAPACKE_sgebal_work* and *LAPACKE_zgebal_work*.

4.19.1257 LAPACKE_sgebrd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgebrd(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, float *a, armpl_int_t lda, float *d,
                           float *e, float *tauq, float *taup);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgebrd*, *LAPACKE_dgebrd*, *LAPACKE_sgebrd* and *LAPACKE_zgebrd*. It also exists with a native Fortran interface as *sgebrd*.

4.19.1258 LAPACKE_sgebrd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgebrd_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, float *a, armpl_int_t lda,
                                float *d, float *e, float *tauq, float *taup,
                                float *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgebrd_work*, *LAPACKE_dgebrd_work*, *LAPACKE_sgebrd_work* and *LAPACKE_zgebrd_work*.

4.19.1259 LAPACKE_sgecon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgecon(armpl_int_t matrix_layout, char norm,
                           armpl_int_t n, const float *a, armpl_int_t lda,
                           float anorm, float *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgecon*, *LAPACKE_dgecon*, *LAPACKE_sgecon* and *LAPACKE_zgecon*. It also exists with a native Fortran interface as *sgecon*.

4.19.1260 LAPACKE_sgecon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgecon_work(armpl_int_t matrix_layout, char norm,
                                armpl_int_t n, const float *a,
                                armpl_int_t lda, float anorm, float *rcond,
                                float *work, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgecon_work*, *LAPACKE_dgecon_work*, *LAPACKE_sgecon_work* and *LAPACKE_zgecon_work*.

4.19.1261 LAPACKE_sgeequ

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgeequ(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, const float *a, armpl_int_t lda,
                           float *r, float *c, float *rowcnd, float *colcnd,
                           float *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeequ*, *LAPACKE_dgeequ*, *LAPACKE_sgeequ* and *LAPACKE_zgeequ*. It also exists with a native Fortran interface as *sgeequ*.

4.19.1262 LAPACKE_sgeequ_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgeequ_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, const float *a,
                                armpl_int_t lda, float *r, float *c,
                                float *rowcnd, float *colcnd, float *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeequ_work*, *LAPACKE_dgeequ_work*, *LAPACKE_sgeequ_work* and *LAPACKE_zgeequ_work*.

4.19.1263 LAPACKE_sgeequb

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgeequb(armpl_int_t matrix_layout, armpl_int_t m,
                            armpl_int_t n, const float *a, armpl_int_t lda,
                            float *r, float *c, float *rowcnd, float *colcnd,
                            float *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeequb*, *LAPACKE_dgeequb*, *LAPACKE_sgeequb* and *LAPACKE_zgeequb*. It also exists with a native Fortran interface as *sgeequb*.

4.19.1264 LAPACKE_sgeequb_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgeequb_work(armpl_int_t matrix_layout, armpl_int_t m,
                                 armpl_int_t n, const float *a,
                                 armpl_int_t lda, float *r, float *c,
                                 float *rowcnd, float *colcnd, float *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.1265 LAPACKE_sgees

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgees(armpl_int_t matrix_layout, char jobvs, char sort,
                          LAPACK_S_SELECT2 select, armpl_int_t n, float *a,
                          armpl_int_t lda, armpl_int_t *sdim, float *wr,
                          float *wi, float *vs, armpl_int_t ldvs);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgees*, *LAPACKE_dgees*, *LAPACKE_sgees* and *LAPACKE_zgees*. It also exists with a native Fortran interface as *sgees*.

4.19.1266 LAPACKE_sgees_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgees_work(armpl_int_t matrix_layout, char jobvs,
                               char sort, LAPACK_S_SELECT2 select,
                               armpl_int_t n, float *a, armpl_int_t lda,
                               armpl_int_t *sdim, float *wr, float *wi,
                               float *vs, armpl_int_t ldvs, float *work,
                               armpl_int_t lwork, armpl_int_t *bwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgees_work*, *LAPACKE_dgees_work*, *LAPACKE_sgees_work* and *LAPACKE_zgees_work*.

4.19.1267 LAPACKE_sgeesx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgeesx(armpl_int_t matrix_layout, char jobvs, char sort,
                           LAPACK_S_SELECT2 select, char sense, armpl_int_t n,
                           float *a, armpl_int_t lda, armpl_int_t *sdim,
```

(continues on next page)

(continued from previous page)

```
                                float *wr, float *wi, float *vs, armpl_int_t ldvs,
                                float *rconde, float *rcondv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeesx*, *LAPACKE_dgeesx*, *LAPACKE_sgeesx* and *LAPACKE_zgeesx*. It also exists with a native Fortran interface as *sgeesx*.

4.19.1268 LAPACKE_sgeesx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgeesx_work(armpl_int_t matrix_layout, char jobvs,
                                char sort, LAPACK_S_SELECT2 select,
                                char sense, armpl_int_t n, float *a,
                                armpl_int_t lda, armpl_int_t *sdim, float *wr,
                                float *wi, float *vs, armpl_int_t ldvs,
                                float *rconde, float *rcondv, float *work,
                                armpl_int_t lwork, armpl_int_t *iwork,
                                armpl_int_t liwork, armpl_int_t *bwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeesx_work*, *LAPACKE_dgeesx_work*, *LAPACKE_sgeesx_work* and *LAPACKE_zgeesx_work*.

4.19.1269 LAPACKE_sgeev

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgeev(armpl_int_t matrix_layout, char jobvl, char jobvr,
                          armpl_int_t n, float *a, armpl_int_t lda, float *wr,
                          float *wi, float *vl, armpl_int_t ldvl, float *vr,
                          armpl_int_t ldvr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeev*, *LAPACKE_dgeev*, *LAPACKE_sgeev* and *LAPACKE_zgeev*. It also exists with a native Fortran interface as *sgeev*.

4.19.1270 LAPACKE_sgeev_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgeev_work(armpl_int_t matrix_layout, char jobvl,
                               char jobvr, armpl_int_t n, float *a,
                               armpl_int_t lda, float *wr, float *wi,
                               float *vl, armpl_int_t ldvl, float *vr,
                               armpl_int_t ldvr, float *work,
                               armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeev_work*, *LAPACKE_dgeev_work*, *LAPACKE_sgeev_work* and *LAPACKE_zgeev_work*.

4.19.1271 LAPACKE_sgeevx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgeevx(armpl_int_t matrix_layout, char balanc, char jobvl,
                           char jobvr, char sense, armpl_int_t n, float *a,
                           armpl_int_t lda, float *wr, float *wi, float *vl,
                           armpl_int_t ldvl, float *vr, armpl_int_t ldvr,
                           armpl_int_t *ilo, armpl_int_t *ihi, float *scale,
                           float *abnrm, float *rconde, float *rcondv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeevx*, *LAPACKE_dgeevx*, *LAPACKE_sgeevx* and *LAPACKE_zgeevx*. It also exists with a native Fortran interface as *sgeevx*.

4.19.1272 LAPACKE_sgeevx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgeevx_work(armpl_int_t matrix_layout, char balanc,
                                char jobvl, char jobvr, char sense,
                                armpl_int_t n, float *a, armpl_int_t lda,
                                float *wr, float *wi, float *vl,
                                armpl_int_t ldvl, float *vr, armpl_int_t ldvr,
                                armpl_int_t *ilo, armpl_int_t *ihi,
                                float *scale, float *abnrm, float *rconde,
                                float *rcondv, float *work, armpl_int_t lwork,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeevx_work*, *LAPACKE_dgeevx_work*, *LAPACKE_sgeevx_work* and *LAPACKE_zgeevx_work*.

4.19.1273 LAPACKE_sgehrd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgehrd(armpl_int_t matrix_layout, armpl_int_t n,
                           armpl_int_t ilo, armpl_int_t ihi, float *a,
                           armpl_int_t lda, float *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgehrd*, *LAPACKE_dgehrd*, *LAPACKE_sgehrd* and *LAPACKE_zgehrd*. It also exists with a native Fortran interface as *sgehrd*.

4.19.1274 LAPACKE_sgehrd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgehrd_work(armpl_int_t matrix_layout, armpl_int_t n,
                                armpl_int_t ilo, armpl_int_t ihi, float *a,
                                armpl_int_t lda, float *tau, float *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgehrd_work*, *LAPACKE_dgehrd_work*, *LAPACKE_sgehrd_work* and *LAPACKE_zgehrd_work*.

4.19.1275 LAPACKE_sgejsv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgejsv(armpl_int_t matrix_layout, char joba, char jobu,
                           char jobv, char jobr, char jobt, char jobp,
                           armpl_int_t m, armpl_int_t n, float *a,
                           armpl_int_t lda, float *sva, float *u,
                           armpl_int_t ldu, float *v, armpl_int_t ldv,
                           float *stat, armpl_int_t *istat);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgejsv*, *LAPACKE_dgejsv*, *LAPACKE_sgejsv* and *LAPACKE_zgejsv*. It also exists with a native Fortran interface as *sgejsv*.

4.19.1276 LAPACKE_sgejsv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgejsv_work(armpl_int_t matrix_layout, char joba,
                                char jobu, char jobv, char jobr, char jobt,
                                char jobp, armpl_int_t m, armpl_int_t n,
                                float *a, armpl_int_t lda, float *sva,
                                float *u, armpl_int_t ldu, float *v,
                                armpl_int_t ldv, float *work,
                                armpl_int_t lwork, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgejsv_work*, *LAPACKE_dgejsv_work*, *LAPACKE_sgejsv_work* and *LAPACKE_zgejsv_work*.

4.19.1277 LAPACKE_sgelq

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgelq(armpl_int_t matrix_layout, armpl_int_t m,
                          armpl_int_t n, float *a, armpl_int_t lda, float *t,
                          armpl_int_t tsize);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelq*, *LAPACKE_dgelq*, *LAPACKE_sgelq* and *LAPACKE_zgelq*. It also exists with a native Fortran interface as *sgelq*.

4.19.1278 LAPACKE_sgelq2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgelq2(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, float *a, armpl_int_t lda,
                           float *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelq2*, *LAPACKE_dgelq2*, *LAPACKE_sgelq2* and *LAPACKE_zgelq2*. It also exists with a native Fortran interface as *sgelq2*.

4.19.1279 LAPACKE_sgelq2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgelq2_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, float *a, armpl_int_t lda,
                                float *tau, float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelq2_work*, *LAPACKE_dgelq2_work*, *LAPACKE_sgelq2_work* and *LAPACKE_zgelq2_work*.

4.19.1280 LAPACKE_sgelq_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgelq_work(armpl_int_t matrix_layout, armpl_int_t m,
                               armpl_int_t n, float *a, armpl_int_t lda,
                               float *t, armpl_int_t tsize, float *work,
                               armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelq_work*, *LAPACKE_dgelq_work*, *LAPACKE_sgelq_work* and *LAPACKE_zgelq_work*.

4.19.1281 LAPACKE_sgelqf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgelqf(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, float *a, armpl_int_t lda,
                           float *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelqf*, *LAPACKE_dgelqf*, *LAPACKE_sgelqf* and *LAPACKE_zgelqf*. It also exists with a native Fortran interface as *sgelqf*.

4.19.1282 LAPACKE_sgelqf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgelqf_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, float *a, armpl_int_t lda,
                                float *tau, float *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelqf_work*, *LAPACKE_dgelqf_work*, *LAPACKE_sgelqf_work* and *LAPACKE_zgelqf_work*.

4.19.1283 LAPACKE_sgels

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgels(armpl_int_t matrix_layout, char trans,
                          armpl_int_t m, armpl_int_t n, armpl_int_t nrhs,
                          float *a, armpl_int_t lda, float *b,
                          armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgels*, *LAPACKE_dgels*, *LAPACKE_sgels* and *LAPACKE_zgels*. It also exists with a native Fortran interface as *sgels*.

4.19.1284 LAPACKE_sgels_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgels_work(armpl_int_t matrix_layout, char trans,
                               armpl_int_t m, armpl_int_t n, armpl_int_t nrhs,
                               float *a, armpl_int_t lda, float *b,
                               armpl_int_t ldb, float *work,
                               armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgels_work*, *LAPACKE_dgels_work*, *LAPACKE_sgels_work* and *LAPACKE_zgels_work*.

4.19.1285 LAPACKE_sgelsd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgelsd(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t nrhs, float *a,
                           armpl_int_t lda, float *b, armpl_int_t ldb,
                           float *s, float rcond, armpl_int_t *rank);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelsd*, *LAPACKE_dgelsd*, *LAPACKE_sgelsd* and *LAPACKE_zgelsd*. It also exists with a native Fortran interface as *sgelsd*.

4.19.1286 LAPACKE_sgelsd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgelsd_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t nrhs, float *a,
                                armpl_int_t lda, float *b, armpl_int_t ldb,
                                float *s, float rcond, armpl_int_t *rank,
                                float *work, armpl_int_t lwork,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelsd_work*, *LAPACKE_dgelsd_work*, *LAPACKE_sgelsd_work* and *LAPACKE_zgelsd_work*.

4.19.1287 LAPACKE_sgelss

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgelss(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t nrhs, float *a,
                           armpl_int_t lda, float *b, armpl_int_t ldb,
                           float *s, float rcond, armpl_int_t *rank);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelss*, *LAPACKE_dgelss*, *LAPACKE_sgelss* and *LAPACKE_zgelss*. It also exists with a native Fortran interface as *sgelss*.

4.19.1288 LAPACKE_sgelss_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgelss_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t nrhs, float *a,
                                armpl_int_t lda, float *b, armpl_int_t ldb,
                                float *s, float rcond, armpl_int_t *rank,
                                float *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelss_work*, *LAPACKE_dgelss_work*, *LAPACKE_sgelss_work* and *LAPACKE_zgelss_work*.

4.19.1289 LAPACKE_sgelsy

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgelsy(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t nrhs, float *a,
                           armpl_int_t lda, float *b, armpl_int_t ldb,
                           armpl_int_t *jpvt, float rcond,
                           armpl_int_t *rank);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelsy*, *LAPACKE_dgelsy*, *LAPACKE_sgelsy* and *LAPACKE_zgelsy*. It also exists with a native Fortran interface as *sgelsy*.

4.19.1290 LAPACKE_sgelsy_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgelsy_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t nrhs, float *a,
                                armpl_int_t lda, float *b, armpl_int_t ldb,
                                armpl_int_t *jpvt, float rcond,
```

(continues on next page)

(continued from previous page)

```
                                armpl_int_t *rank, float *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelsy_work*, *LAPACKE_dgelsy_work*, *LAPACKE_sgelsy_work* and *LAPACKE_zgelsy_work*.

4.19.1291 LAPACKE_sgemlq

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgemlq(armpl_int_t matrix_layout, char side, char trans,
                           armpl_int_t m, armpl_int_t n, armpl_int_t k,
                           const float *a, armpl_int_t lda, const float *t,
                           armpl_int_t tsize, float *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgemlq*, *LAPACKE_dgemlq*, *LAPACKE_sgemlq* and *LAPACKE_zgemlq*. It also exists with a native Fortran interface as *sgemlq*.

4.19.1292 LAPACKE_sgemlq_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgemlq_work(armpl_int_t matrix_layout, char side,
                                char trans, armpl_int_t m, armpl_int_t n,
                                armpl_int_t k, const float *a,
                                armpl_int_t lda, const float *t,
                                armpl_int_t tsize, float *c, armpl_int_t ldc,
                                float *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgemlq_work*, *LAPACKE_dgemlq_work*, *LAPACKE_sgemlq_work* and *LAPACKE_zgemlq_work*.

4.19.1293 LAPACKE_sgemqr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgemqr(armpl_int_t matrix_layout, char side, char trans,
                           armpl_int_t m, armpl_int_t n, armpl_int_t k,
                           const float *a, armpl_int_t lda, const float *t,
                           armpl_int_t tsize, float *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgemqr*, *LAPACKE_dgemqr*, *LAPACKE_sgemqr* and *LAPACKE_zgemqr*. It also exists with a native Fortran interface as *sgemqr*.

4.19.1294 LAPACKE_sgemqr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgemqr_work(armpl_int_t matrix_layout, char side,
                                char trans, armpl_int_t m, armpl_int_t n,
                                armpl_int_t k, const float *a,
                                armpl_int_t lda, const float *t,
                                armpl_int_t tsize, float *c, armpl_int_t ldc,
                                float *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgemqr_work*, *LAPACKE_dgemqr_work*, *LAPACKE_sgemqr_work* and *LAPACKE_zgemqr_work*.

4.19.1295 LAPACKE_sgemqrt

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgemqrt(armpl_int_t matrix_layout, char side, char trans,
                            armpl_int_t m, armpl_int_t n, armpl_int_t k,
                            armpl_int_t nb, const float *v, armpl_int_t ldv,
                            const float *t, armpl_int_t ldt, float *c,
                            armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgemqrt*, *LAPACKE_dgemqrt*, *LAPACKE_sgemqrt* and *LAPACKE_zgemqrt*. It also exists with a native Fortran interface as *sgemqrt*.

4.19.1296 LAPACKE_sgemqrt_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgemqrt_work(armpl_int_t matrix_layout, char side,
                                 char trans, armpl_int_t m, armpl_int_t n,
                                 armpl_int_t k, armpl_int_t nb,
                                 const float *v, armpl_int_t ldv,
                                 const float *t, armpl_int_t ldt, float *c,
                                 armpl_int_t ldc, float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgemqrt_work*, *LAPACKE_dgemqrt_work*, *LAPACKE_sgemqrt_work* and *LAPACKE_zgemqrt_work*.

4.19.1297 LAPACKE_sgeqlf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgeqlf(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, float *a, armpl_int_t lda,
                           float *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqlf*, *LAPACKE_dgeqlf*, *LAPACKE_sgeqlf* and *LAPACKE_zgeqlf*. It also exists with a native Fortran interface as *sgeqlf*.

4.19.1298 LAPACKE_sgeqlf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgeqlf_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, float *a, armpl_int_t lda,
                                float *tau, float *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqlf_work*, *LAPACKE_dgeqlf_work*, *LAPACKE_sgeqlf_work* and *LAPACKE_zgeqlf_work*.

4.19.1299 LAPACKE_sgeqp3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgeqp3(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, float *a, armpl_int_t lda,
                           armpl_int_t *jpvt, float *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqp3*, *LAPACKE_dgeqp3*, *LAPACKE_sgeqp3* and *LAPACKE_zgeqp3*. It also exists with a native Fortran interface as *sgeqp3*.

4.19.1300 LAPACKE_sgeqp3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgeqp3_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, float *a, armpl_int_t lda,
                                armpl_int_t *jpvt, float *tau, float *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqp3_work*, *LAPACKE_dgeqp3_work*, *LAPACKE_sgeqp3_work* and *LAPACKE_zgeqp3_work*.

4.19.1301 LAPACKE_sgeqpf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgeqpf(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, float *a, armpl_int_t lda,
                           armpl_int_t *jpvt, float *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqpf*, *LAPACKE_dgeqpf*, *LAPACKE_sgeqpf* and *LAPACKE_zgeqpf*.

4.19.1302 LAPACKE_sgeqpf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgeqpf_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, float *a, armpl_int_t lda,
                                armpl_int_t *jpvt, float *tau, float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqpf_work*, *LAPACKE_dgeqpf_work*, *LAPACKE_sgeqpf_work* and *LAPACKE_zgeqpf_work*.

4.19.1303 LAPACKE_sgeqr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgeqr(armpl_int_t matrix_layout, armpl_int_t m,
                          armpl_int_t n, float *a, armpl_int_t lda, float *t,
                          armpl_int_t tsize);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqr*, *LAPACKE_dgeqr*, *LAPACKE_sgeqr* and *LAPACKE_zgeqr*. It also exists with a native Fortran interface as *sgeqr*.

4.19.1304 LAPACKE_sgeqr2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgeqr2(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, float *a, armpl_int_t lda,
                           float *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqr2*, *LAPACKE_dgeqr2*, *LAPACKE_sgeqr2* and *LAPACKE_zgeqr2*. It also exists with a native Fortran interface as *sgeqr2*.

4.19.1305 LAPACKE_sgeqr2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgeqr2_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, float *a, armpl_int_t lda,
                                float *tau, float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqr2_work*, *LAPACKE_dgeqr2_work*, *LAPACKE_sgeqr2_work* and *LAPACKE_zgeqr2_work*.

4.19.1306 LAPACKE_sgeqr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgeqr_work(armpl_int_t matrix_layout, armpl_int_t m,
                               armpl_int_t n, float *a, armpl_int_t lda,
                               float *t, armpl_int_t tsize, float *work,
                               armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqr_work*, *LAPACKE_dgeqr_work*, *LAPACKE_sgeqr_work* and *LAPACKE_zgeqr_work*.

4.19.1307 LAPACKE_sgeqrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgeqrf(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, float *a, armpl_int_t lda,
                           float *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrf*, *LAPACKE_dgeqrf*, *LAPACKE_sgeqrf* and *LAPACKE_zgeqrf*. It also exists with a native Fortran interface as *sgeqrf*.

4.19.1308 LAPACKE_sgeqrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgeqrf_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, float *a, armpl_int_t lda,
                                float *tau, float *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrf_work*, *LAPACKE_dgeqrf_work*, *LAPACKE_sgeqrf_work* and *LAPACKE_zgeqrf_work*.

4.19.1309 LAPACKE_sgeqrfp

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgeqrfp(armpl_int_t matrix_layout, armpl_int_t m,
                            armpl_int_t n, float *a, armpl_int_t lda,
                            float *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrfp*, *LAPACKE_dgeqrfp*, *LAPACKE_sgeqrfp* and *LAPACKE_zgeqrfp*. It also exists with a native Fortran interface as *sgeqrfp*.

4.19.1310 LAPACKE_sgeqrfp_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgeqrfp_work(armpl_int_t matrix_layout, armpl_int_t m,
                                 armpl_int_t n, float *a, armpl_int_t lda,
                                 float *tau, float *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrfp_work*, *LAPACKE_dgeqrfp_work*, *LAPACKE_sgeqrfp_work* and *LAPACKE_zgeqrfp_work*.

4.19.1311 LAPACKE_sgeqrt

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgeqrt(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t nb, float *a,
                           armpl_int_t lda, float *t, armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrt*, *LAPACKE_dgeqrt*, *LAPACKE_sgeqrt* and *LAPACKE_zgeqrt*. It also exists with a native Fortran interface as *sgeqrt*.

4.19.1312 LAPACKE_sgeqrt2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgeqrt2(armpl_int_t matrix_layout, armpl_int_t m,
                            armpl_int_t n, float *a, armpl_int_t lda,
                            float *t, armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrt2*, *LAPACKE_dgeqrt2*, *LAPACKE_sgeqrt2* and *LAPACKE_zgeqrt2*. It also exists with a native Fortran interface as *sgeqrt2*.

4.19.1313 LAPACKE_sgeqrt2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgeqrt2_work(armpl_int_t matrix_layout, armpl_int_t m,
                                 armpl_int_t n, float *a, armpl_int_t lda,
                                 float *t, armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrt2_work*, *LAPACKE_dgeqrt2_work*, *LAPACKE_sgeqrt2_work* and *LAPACKE_zgeqrt2_work*.

4.19.1314 LAPACKE_sgeqrt3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgeqrt3(armpl_int_t matrix_layout, armpl_int_t m,
                            armpl_int_t n, float *a, armpl_int_t lda,
                            float *t, armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrt3*, *LAPACKE_dgeqrt3*, *LAPACKE_sgeqrt3* and *LAPACKE_zgeqrt3*. It also exists with a native Fortran interface as *sgeqrt3*.

4.19.1315 LAPACKE_sgeqrt3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgeqrt3_work(armpl_int_t matrix_layout, armpl_int_t m,
                                 armpl_int_t n, float *a, armpl_int_t lda,
                                 float *t, armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrt3_work*, *LAPACKE_dgeqrt3_work*, *LAPACKE_sgeqrt3_work* and *LAPACKE_zgeqrt3_work*.

4.19.1316 LAPACKE_sgeqrt_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgeqrt_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t nb, float *a,
                                armpl_int_t lda, float *t, armpl_int_t ldt,
                                float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.1317 LAPACKE_sgerfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgerfs(armpl_int_t matrix_layout, char trans,
                           armpl_int_t n, armpl_int_t nrhs, const float *a,
                           armpl_int_t lda, const float *af, armpl_int_t ldaf,
                           const armpl_int_t *ipiv, const float *b,
                           armpl_int_t ldb, float *x, armpl_int_t ldx,
                           float *ferr, float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgerfs*, *LAPACKE_dgerfs*, *LAPACKE_sgerfs* and *LAPACKE_zgerfs*. It also exists with a native Fortran interface as *sgerfs*.

4.19.1318 LAPACKE_sgerfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgerfs_work(armpl_int_t matrix_layout, char trans,
                                armpl_int_t n, armpl_int_t nrhs,
                                const float *a, armpl_int_t lda,
                                const float *af, armpl_int_t ldaf,
                                const armpl_int_t *ipiv, const float *b,
                                armpl_int_t ldb, float *x, armpl_int_t ldx,
                                float *ferr, float *berr, float *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgerfs_work*, *LAPACKE_dgerfs_work*, *LAPACKE_sgerfs_work* and *LAPACKE_zgerfs_work*.

4.19.1319 LAPACKE_sgerfsx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgerfsx(armpl_int_t matrix_layout, char trans, char equed,
                            armpl_int_t n, armpl_int_t nrhs, const float *a,
                            armpl_int_t lda, const float *af,
                            armpl_int_t ldaf, const armpl_int_t *ipiv,
                            const float *r, const float *c, const float *b,
                            armpl_int_t ldb, float *x, armpl_int_t ldx,
                            float *rcond, float *berr, armpl_int_t n_err_bnds,
                            float *err_bnds_norm, float *err_bnds_comp,
                            armpl_int_t nparams, float *params);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgerfsx*, *LAPACKE_dgerfsx*, *LAPACKE_sgerfsx* and *LAPACKE_zgerfsx*. It also exists with a native Fortran interface as *sgerfsx*.

4.19.1320 LAPACKE_sgerfsx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgerfsx_work(armpl_int_t matrix_layout, char trans,
                                 char equed, armpl_int_t n, armpl_int_t nrhs,
                                 const float *a, armpl_int_t lda,
                                 const float *af, armpl_int_t ldaf,
                                 const armpl_int_t *ipiv, const float *r,
                                 const float *c, const float *b,
                                 armpl_int_t ldb, float *x, armpl_int_t ldx,
                                 float *rcond, float *berr,
                                 armpl_int_t n_err_bnds, float *err_bnds_norm,
                                 float *err_bnds_comp, armpl_int_t nparams,
                                 float *params, float *work,
                                 armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgerfsx_work*, *LAPACKE_dgerfsx_work*, *LAPACKE_sgerfsx_work* and *LAPACKE_zgerfsx_work*.

4.19.1321 LAPACKE_sgerqf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgerqf(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, float *a, armpl_int_t lda,
                           float *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgerqf*, *LAPACKE_dgerqf*, *LAPACKE_sgerqf* and *LAPACKE_zgerqf*. It also exists with a native Fortran interface as *sgerqf*.

4.19.1322 LAPACKE_sgerqf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgerqf_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, float *a, armpl_int_t lda,
                                float *tau, float *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgerqf_work*, *LAPACKE_dgerqf_work*, *LAPACKE_sgerqf_work* and *LAPACKE_zgerqf_work*.

4.19.1323 LAPACKE_sgesdd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgesdd(armpl_int_t matrix_layout, char jobz,
                           armpl_int_t m, armpl_int_t n, float *a,
                           armpl_int_t lda, float *s, float *u,
                           armpl_int_t ldu, float *vt, armpl_int_t ldvt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesdd*, *LAPACKE_dgesdd*, *LAPACKE_sgesdd* and *LAPACKE_zgesdd*. It also exists with a native Fortran interface as *sgesdd*.

4.19.1324 LAPACKE_sgesdd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgesdd_work(armpl_int_t matrix_layout, char jobz,
                                armpl_int_t m, armpl_int_t n, float *a,
                                armpl_int_t lda, float *s, float *u,
                                armpl_int_t ldu, float *vt, armpl_int_t ldvt,
                                float *work, armpl_int_t lwork,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesdd_work*, *LAPACKE_dgesdd_work*, *LAPACKE_sgesdd_work* and *LAPACKE_zgesdd_work*.

4.19.1325 LAPACKE_sgesv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgesv(armpl_int_t matrix_layout, armpl_int_t n,
                          armpl_int_t nrhs, float *a, armpl_int_t lda,
                          armpl_int_t *ipiv, float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesv*, *LAPACKE_dgesv*, *LAPACKE_sgesv* and *LAPACKE_zgesv*. It also exists with a native Fortran interface as *sgesv*.

4.19.1326 LAPACKE_sgesv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgesv_work(armpl_int_t matrix_layout, armpl_int_t n,
                               armpl_int_t nrhs, float *a, armpl_int_t lda,
                               armpl_int_t *ipiv, float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesv_work*, *LAPACKE_dgesv_work*, *LAPACKE_sgesv_work* and *LAPACKE_zgesv_work*.

4.19.1327 LAPACKE_sgesvd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgesvd(armpl_int_t matrix_layout, char jobu, char jobvt,
                           armpl_int_t m, armpl_int_t n, float *a,
                           armpl_int_t lda, float *s, float *u,
                           armpl_int_t ldu, float *vt, armpl_int_t ldvt,
                           float *superb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvd*, *LAPACKE_dgesvd*, *LAPACKE_sgesvd* and *LAPACKE_zgesvd*. It also exists with a native Fortran interface as *sgesvd*.

4.19.1328 LAPACKE_sgesvd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgesvd_work(armpl_int_t matrix_layout, char jobu,
                                char jobvt, armpl_int_t m, armpl_int_t n,
                                float *a, armpl_int_t lda, float *s, float *u,
                                armpl_int_t ldu, float *vt, armpl_int_t ldvt,
                                float *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvd_work*, *LAPACKE_dgesvd_work*, *LAPACKE_sgesvd_work* and *LAPACKE_zgesvd_work*.

4.19.1329 LAPACKE_sgesvdx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgesvdx(armpl_int_t matrix_layout, char jobu, char jobvt,
                            char range, armpl_int_t m, armpl_int_t n,
                            float *a, armpl_int_t lda, float vl, float vu,
                            armpl_int_t il, armpl_int_t iu, armpl_int_t *ns,
                            float *s, float *u, armpl_int_t ldu, float *vt,
                            armpl_int_t ldvt, armpl_int_t *superb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvdx*, *LAPACKE_dgesvdx*, *LAPACKE_sgesvdx* and *LAPACKE_zgesvdx*. It also exists with a native Fortran interface as *sgesvdx*.

4.19.1330 LAPACKE_sgesvdx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgesvdx_work(armpl_int_t matrix_layout, char jobu,
                                 char jobvt, char range, armpl_int_t m,
                                 armpl_int_t n, float *a, armpl_int_t lda,
                                 float vl, float vu, armpl_int_t il,
                                 armpl_int_t iu, armpl_int_t *ns, float *s,
                                 float *u, armpl_int_t ldu, float *vt,
                                 armpl_int_t ldvt, float *work,
                                 armpl_int_t lwork, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvdx_work*, *LAPACKE_dgesvdx_work*, *LAPACKE_sgesvdx_work* and *LAPACKE_zgesvdx_work*.

4.19.1331 LAPACKE_sgesvj

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgesvj(armpl_int_t matrix_layout, char joba, char jobu,
                           char jobv, armpl_int_t m, armpl_int_t n, float *a,
                           armpl_int_t lda, float *sva, armpl_int_t mv,
                           float *v, armpl_int_t ldv, float *stat);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvj*, *LAPACKE_dgesvj*, *LAPACKE_sgesvj* and *LAPACKE_zgesvj*. It also exists with a native Fortran interface as *sgesvj*.

4.19.1332 LAPACKE_sgesvj_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgesvj_work(armpl_int_t matrix_layout, char joba,
                                char jobu, char jobv, armpl_int_t m,
                                armpl_int_t n, float *a, armpl_int_t lda,
                                float *sva, armpl_int_t mv, float *v,
                                armpl_int_t ldv, float *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvj_work*, *LAPACKE_dgesvj_work*, *LAPACKE_sgesvj_work* and *LAPACKE_zgesvj_work*.

4.19.1333 LAPACKE_sgesvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgesvx(armpl_int_t matrix_layout, char fact, char trans,
                           armpl_int_t n, armpl_int_t nrhs, float *a,
                           armpl_int_t lda, float *af, armpl_int_t ldaf,
                           armpl_int_t *ipiv, char *equed, float *r, float *c,
                           float *b, armpl_int_t ldb, float *x,
                           armpl_int_t ldx, float *rcond, float *ferr,
                           float *berr, float *rpivot);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvx*, *LAPACKE_dgesvx*, *LAPACKE_sgesvx* and *LAPACKE_zgesvx*. It also exists with a native Fortran interface as *sgesvx*.

4.19.1334 LAPACKE_sgesvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgesvx_work(armpl_int_t matrix_layout, char fact,
                                char trans, armpl_int_t n, armpl_int_t nrhs,
                                float *a, armpl_int_t lda, float *af,
                                armpl_int_t ldaf, armpl_int_t *ipiv,
                                char *equed, float *r, float *c, float *b,
                                armpl_int_t ldb, float *x, armpl_int_t ldx,
                                float *rcond, float *ferr, float *berr,
                                float *work, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvx_work*, *LAPACKE_dgesvx_work*, *LAPACKE_sgesvx_work* and *LAPACKE_zgesvx_work*.

4.19.1335 LAPACKE_sgesvxx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgesvxx(armpl_int_t matrix_layout, char fact, char trans,
                            armpl_int_t n, armpl_int_t nrhs, float *a,
                            armpl_int_t lda, float *af, armpl_int_t ldaf,
                            armpl_int_t *ipiv, char *equed, float *r,
                            float *c, float *b, armpl_int_t ldb, float *x,
                            armpl_int_t ldx, float *rcond, float *rpvgrw,
                            float *berr, armpl_int_t n_err_bnds,
                            float *err_bnds_norm, float *err_bnds_comp,
                            armpl_int_t nparams, float *params);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvxx*, *LAPACKE_dgesvxx*, *LAPACKE_sgesvxx* and *LAPACKE_zgesvxx*. It also exists with a native Fortran interface as *sgesvxx*.

4.19.1336 LAPACKE_sgesvxx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgesvxx_work(armpl_int_t matrix_layout, char fact,
                                 char trans, armpl_int_t n, armpl_int_t nrhs,
                                 float *a, armpl_int_t lda, float *af,
                                 armpl_int_t ldaf, armpl_int_t *ipiv,
                                 char *equed, float *r, float *c, float *b,
                                 armpl_int_t ldb, float *x, armpl_int_t ldx,
```

(continues on next page)

(continued from previous page)

```
                                    float *rcond, float *rpvgrw, float *berr,
                                    armpl_int_t n_err_bnds, float *err_bnds_norm,
                                    float *err_bnds_comp, armpl_int_t nparams,
                                    float *params, float *work,
                                    armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvxx_work*, *LAPACKE_dgesvxx_work*, *LAPACKE_sgesvxx_work* and *LAPACKE_zgesvxx_work*.

4.19.1337 LAPACKE_sgetf2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgetf2(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, float *a, armpl_int_t lda,
                           armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetf2*, *LAPACKE_dgetf2*, *LAPACKE_sgetf2* and *LAPACKE_zgetf2*. It also exists with a native Fortran interface as *sgetf2*.

4.19.1338 LAPACKE_sgetf2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgetf2_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, float *a, armpl_int_t lda,
                                armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetf2_work*, *LAPACKE_dgetf2_work*, *LAPACKE_sgetf2_work* and *LAPACKE_zgetf2_work*.

4.19.1339 LAPACKE_sgetrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgetrf(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, float *a, armpl_int_t lda,
                           armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetrf*, *LAPACKE_dgetrf*, *LAPACKE_sgetrf* and *LAPACKE_zgetrf*. It also exists with a native Fortran interface as *sgetrf*.

4.19.1340 LAPACKE_sgetrf2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgetrf2(armpl_int_t matrix_layout, armpl_int_t m,
                            armpl_int_t n, float *a, armpl_int_t lda,
                            armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetrf2*, *LAPACKE_dgetrf2*, *LAPACKE_sgetrf2* and *LAPACKE_zgetrf2*. It also exists with a native Fortran interface as *sgetrf2*.

4.19.1341 LAPACKE_sgetrf2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgetrf2_work(armpl_int_t matrix_layout, armpl_int_t m,
                                 armpl_int_t n, float *a, armpl_int_t lda,
                                 armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetrf2_work*, *LAPACKE_dgetrf2_work*, *LAPACKE_sgetrf2_work* and *LAPACKE_zgetrf2_work*.

4.19.1342 LAPACKE_sgetrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgetrf_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, float *a, armpl_int_t lda,
                                armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetrf_work*, *LAPACKE_dgetrf_work*, *LAPACKE_sgetrf_work* and *LAPACKE_zgetrf_work*.

4.19.1343 LAPACKE_sgetri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgetri(armpl_int_t matrix_layout, armpl_int_t n, float *a,
                           armpl_int_t lda, const armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetri*, *LAPACKE_dgetri*, *LAPACKE_sgetri* and *LAPACKE_zgetri*. It also exists with a native Fortran interface as *sgetri*.

4.19.1344 LAPACKE_sgetri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgetri_work(armpl_int_t matrix_layout, armpl_int_t n,
                                float *a, armpl_int_t lda,
                                const armpl_int_t *ipiv, float *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetri_work*, *LAPACKE_dgetri_work*, *LAPACKE_sgetri_work* and *LAPACKE_zgetri_work*.

4.19.1345 LAPACKE_sgetrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgetrs(armpl_int_t matrix_layout, char trans,
                           armpl_int_t n, armpl_int_t nrhs, const float *a,
                           armpl_int_t lda, const armpl_int_t *ipiv, float *b,
                           armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetrs*, *LAPACKE_dgetrs*, *LAPACKE_sgetrs* and *LAPACKE_zgetrs*. It also exists with a native Fortran interface as *sgetrs*.

4.19.1346 LAPACKE_sgetrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgetrs_work(armpl_int_t matrix_layout, char trans,
                                armpl_int_t n, armpl_int_t nrhs,
                                const float *a, armpl_int_t lda,
                                const armpl_int_t *ipiv, float *b,
                                armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetrs_work*, *LAPACKE_dgetrs_work*, *LAPACKE_sgetrs_work* and *LAPACKE_zgetrs_work*.

4.19.1347 LAPACKE_sgetsls

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgetsls(armpl_int_t matrix_layout, char trans,
                            armpl_int_t m, armpl_int_t n, armpl_int_t nrhs,
                            float *a, armpl_int_t lda, float *b,
                            armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetsls*, *LAPACKE_dgetsls*, *LAPACKE_sgetsls* and *LAPACKE_zgetsls*. It also exists with a native Fortran interface as *sgetsls*.

4.19.1348 LAPACKE_sgetsls_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgetsls_work(armpl_int_t matrix_layout, char trans,
                                 armpl_int_t m, armpl_int_t n,
                                 armpl_int_t nrhs, float *a, armpl_int_t lda,
                                 float *b, armpl_int_t ldb, float *work,
                                 armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetsls_work*, *LAPACKE_dgetsls_work*, *LAPACKE_sgetsls_work* and *LAPACKE_zgetsls_work*.

4.19.1349 LAPACKE_sggbak

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sggbak(armpl_int_t matrix_layout, char job, char side,
                           armpl_int_t n, armpl_int_t ilo, armpl_int_t ihi,
                           const float *lscale, const float *rscale,
                           armpl_int_t m, float *v, armpl_int_t ldv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggbak*, *LAPACKE_dggbak*, *LAPACKE_sggbak* and *LAPACKE_zggbak*. It also exists with a native Fortran interface as *sggbak*.

4.19.1350 LAPACKE_sggbak_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sggbak_work(armpl_int_t matrix_layout, char job,
                                char side, armpl_int_t n, armpl_int_t ilo,
                                armpl_int_t ihi, const float *lscale,
                                const float *rscale, armpl_int_t m, float *v,
                                armpl_int_t ldv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggbak_work*, *LAPACKE_dggbak_work*, *LAPACKE_sggbak_work* and *LAPACKE_zggbak_work*.

4.19.1351 LAPACKE_sggbal

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sggbal(armpl_int_t matrix_layout, char job, armpl_int_t n,
                           float *a, armpl_int_t lda, float *b,
                           armpl_int_t ldb, armpl_int_t *ilo,
                           armpl_int_t *ihi, float *lscale, float *rscale);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggbal*, *LAPACKE_dggbal*, *LAPACKE_sggbal* and *LAPACKE_zggbal*. It also exists with a native Fortran interface as *sggbal*.

4.19.1352 LAPACKE_sggbal_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sggbal_work(armpl_int_t matrix_layout, char job,
                                armpl_int_t n, float *a, armpl_int_t lda,
                                float *b, armpl_int_t ldb, armpl_int_t *ilo,
                                armpl_int_t *ihi, float *lscale,
                                float *rscale, float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggbal_work*, *LAPACKE_dggbal_work*, *LAPACKE_sggbal_work* and *LAPACKE_zggbal_work*.

4.19.1353 LAPACKE_sgges

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgges(armpl_int_t matrix_layout, char jobvsl, char jobvsr,
                          char sort, LAPACK_S_SELECT3 selctg, armpl_int_t n,
                          float *a, armpl_int_t lda, float *b,
                          armpl_int_t ldb, armpl_int_t *sdim, float *alphar,
                          float *alphai, float *beta, float *vsl,
                          armpl_int_t ldvsl, float *vsr, armpl_int_t ldvsr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgges*, *LAPACKE_dgges*, *LAPACKE_sgges* and *LAPACKE_zgges*. It also exists with a native Fortran interface as *sgges*.

4.19.1354 LAPACKE_sgges3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgges3(armpl_int_t matrix_layout, char jobvsl,
                           char jobvsr, char sort, LAPACK_S_SELECT3 selctg,
                           armpl_int_t n, float *a, armpl_int_t lda, float *b,
                           armpl_int_t ldb, armpl_int_t *sdim, float *alphar,
                           float *alphai, float *beta, float *vsl,
                           armpl_int_t ldvsl, float *vsr, armpl_int_t ldvsr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgges3*, *LAPACKE_dgges3*, *LAPACKE_sgges3* and *LAPACKE_zgges3*. It also exists with a native Fortran interface as *sgges3*.

4.19.1355 LAPACKE_sgges3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgges3_work(armpl_int_t matrix_layout, char jobvsl,
                                char jobvsr, char sort,
                                LAPACK_S_SELECT3 selctg, armpl_int_t n,
                                float *a, armpl_int_t lda, float *b,
                                armpl_int_t ldb, armpl_int_t *sdim,
                                float *alphar, float *alphai, float *beta,
                                float *vsl, armpl_int_t ldvsl, float *vsr,
                                armpl_int_t ldvsr, float *work,
                                armpl_int_t lwork, armpl_int_t *bwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgges3_work*, *LAPACKE_dgges3_work*, *LAPACKE_sgges3_work* and *LAPACKE_zgges3_work*.

4.19.1356 LAPACKE_sgges_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgges_work(armpl_int_t matrix_layout, char jobvsl,
                               char jobvsr, char sort,
                               LAPACK_S_SELECT3 selctg, armpl_int_t n,
                               float *a, armpl_int_t lda, float *b,
                               armpl_int_t ldb, armpl_int_t *sdim,
                               float *alphar, float *alphai, float *beta,
                               float *vsl, armpl_int_t ldvsl, float *vsr,
                               armpl_int_t ldvsr, float *work,
                               armpl_int_t lwork, armpl_int_t *bwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgges_work*, *LAPACKE_dgges_work*, *LAPACKE_sgges_work* and *LAPACKE_zgges_work*.

4.19.1357 LAPACKE_sggesx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sggesx(armpl_int_t matrix_layout, char jobvsl,
                           char jobvsr, char sort, LAPACK_S_SELECT3 selctg,
                           char sense, armpl_int_t n, float *a,
                           armpl_int_t lda, float *b, armpl_int_t ldb,
                           armpl_int_t *sdim, float *alphar, float *alphai,
                           float *beta, float *vsl, armpl_int_t ldvsl,
```

(continues on next page)

(continued from previous page)

```
                                float *vsr, armpl_int_t ldvsr, float *rconde,
                                float *rcondv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggesx*, *LAPACKE_dggesx*, *LAPACKE_sggesx* and *LAPACKE_zggesx*. It also exists with a native Fortran interface as *sggesx*.

4.19.1358 LAPACKE_sggesx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sggesx_work(armpl_int_t matrix_layout, char jobvsl,
                                char jobvsr, char sort,
                                LAPACK_S_SELECT3 selctg, char sense,
                                armpl_int_t n, float *a, armpl_int_t lda,
                                float *b, armpl_int_t ldb, armpl_int_t *sdim,
                                float *alphar, float *alphai, float *beta,
                                float *vsl, armpl_int_t ldvsl, float *vsr,
                                armpl_int_t ldvsr, float *rconde,
                                float *rcondv, float *work, armpl_int_t lwork,
                                armpl_int_t *iwork, armpl_int_t liwork,
                                armpl_int_t *bwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggesx_work*, *LAPACKE_dggesx_work*, *LAPACKE_sggesx_work* and *LAPACKE_zggesx_work*.

4.19.1359 LAPACKE_sggev

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sggev(armpl_int_t matrix_layout, char jobvl, char jobvr,
                          armpl_int_t n, float *a, armpl_int_t lda, float *b,
                          armpl_int_t ldb, float *alphar, float *alphai,
                          float *beta, float *vl, armpl_int_t ldvl, float *vr,
                          armpl_int_t ldvr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggev*, *LAPACKE_dggev*, *LAPACKE_sggev* and *LAPACKE_zggev*. It also exists with a native Fortran interface as *sggev*.

4.19.1360 LAPACKE_sggev3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sggev3(armpl_int_t matrix_layout, char jobvl, char jobvr,
                           armpl_int_t n, float *a, armpl_int_t lda, float *b,
                           armpl_int_t ldb, float *alphar, float *alphai,
                           float *beta, float *vl, armpl_int_t ldvl,
                           float *vr, armpl_int_t ldvr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggev3*, *LAPACKE_dggev3*, *LAPACKE_sggev3* and *LAPACKE_zggev3*. It also exists with a native Fortran interface as *sggev3*.

4.19.1361 LAPACKE_sggev3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sggev3_work(armpl_int_t matrix_layout, char jobvl,
                                char jobvr, armpl_int_t n, float *a,
                                armpl_int_t lda, float *b, armpl_int_t ldb,
                                float *alphar, float *alphai, float *beta,
                                float *vl, armpl_int_t ldvl, float *vr,
                                armpl_int_t ldvr, float *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggev3_work*, *LAPACKE_dggev3_work*, *LAPACKE_sggev3_work* and *LAPACKE_zggev3_work*.

4.19.1362 LAPACKE_sggev_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sggev_work(armpl_int_t matrix_layout, char jobvl,
                               char jobvr, armpl_int_t n, float *a,
                               armpl_int_t lda, float *b, armpl_int_t ldb,
                               float *alphar, float *alphai, float *beta,
                               float *vl, armpl_int_t ldvl, float *vr,
                               armpl_int_t ldvr, float *work,
                               armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggev_work*, *LAPACKE_dggev_work*, *LAPACKE_sggev_work* and *LAPACKE_zggev_work*.

4.19.1363 LAPACKE_sggevx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sggevx(armpl_int_t matrix_layout, char balanc, char jobvl,
                           char jobvr, char sense, armpl_int_t n, float *a,
                           armpl_int_t lda, float *b, armpl_int_t ldb,
                           float *alphar, float *alphai, float *beta,
                           float *vl, armpl_int_t ldvl, float *vr,
                           armpl_int_t ldvr, armpl_int_t *ilo,
                           armpl_int_t *ihi, float *lscale, float *rscale,
                           float *abnrm, float *bbnrm, float *rconde,
                           float *rcondv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggevx*, *LAPACKE_dggevx*, *LAPACKE_sggevx* and *LAPACKE_zggevx*. It also exists with a native Fortran interface as *sggevx*.

4.19.1364 LAPACKE_sggevx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sggevx_work(armpl_int_t matrix_layout, char balanc,
                                char jobvl, char jobvr, char sense,
                                armpl_int_t n, float *a, armpl_int_t lda,
                                float *b, armpl_int_t ldb, float *alphar,
                                float *alphai, float *beta, float *vl,
                                armpl_int_t ldvl, float *vr, armpl_int_t ldvr,
                                armpl_int_t *ilo, armpl_int_t *ihi,
                                float *lscale, float *rscale, float *abnrm,
                                float *bbnrm, float *rconde, float *rcondv,
                                float *work, armpl_int_t lwork,
                                armpl_int_t *iwork, armpl_int_t *bwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggevx_work*, *LAPACKE_dggevx_work*, *LAPACKE_sggevx_work* and *LAPACKE_zggevx_work*.

4.19.1365 LAPACKE_sggglm

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sggglm(armpl_int_t matrix_layout, armpl_int_t n,
                           armpl_int_t m, armpl_int_t p, float *a,
                           armpl_int_t lda, float *b, armpl_int_t ldb,
                           float *d, float *x, float *y);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggglm*, *LAPACKE_dggglm*, *LAPACKE_sggglm* and *LAPACKE_zggglm*. It also exists with a native Fortran interface as *sggglm*.

4.19.1366 LAPACKE_sggglm_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sggglm_work(armpl_int_t matrix_layout, armpl_int_t n,
                                armpl_int_t m, armpl_int_t p, float *a,
                                armpl_int_t lda, float *b, armpl_int_t ldb,
                                float *d, float *x, float *y, float *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggglm_work*, *LAPACKE_dggglm_work*, *LAPACKE_sggglm_work* and *LAPACKE_zggglm_work*.

4.19.1367 LAPACKE_sgghd3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgghd3(armpl_int_t matrix_layout, char compq, char compz,
                           armpl_int_t n, armpl_int_t ilo, armpl_int_t ihi,
                           float *a, armpl_int_t lda, float *b,
                           armpl_int_t ldb, float *q, armpl_int_t ldq,
                           float *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgghd3*, *LAPACKE_dgghd3*, *LAPACKE_sgghd3* and *LAPACKE_zgghd3*. It also exists with a native Fortran interface as *sgghd3*.

4.19.1368 LAPACKE_sgghd3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgghd3_work(armpl_int_t matrix_layout, char compq,
                                char compz, armpl_int_t n, armpl_int_t ilo,
                                armpl_int_t ihi, float *a, armpl_int_t lda,
                                float *b, armpl_int_t ldb, float *q,
```

(continues on next page)

(continued from previous page)

```
                                armpl_int_t ldq, float *z, armpl_int_t ldz,
                                float *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgghd3_work*, *LAPACKE_dgghd3_work*, *LAPACKE_sgghd3_work* and *LAPACKE_zgghd3_work*.

4.19.1369 LAPACKE_sgghrd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgghrd(armpl_int_t matrix_layout, char compq, char compz,
                           armpl_int_t n, armpl_int_t ilo, armpl_int_t ihi,
                           float *a, armpl_int_t lda, float *b,
                           armpl_int_t ldb, float *q, armpl_int_t ldq,
                           float *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgghrd*, *LAPACKE_dgghrd*, *LAPACKE_sgghrd* and *LAPACKE_zgghrd*. It also exists with a native Fortran interface as *sgghrd*.

4.19.1370 LAPACKE_sgghrd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgghrd_work(armpl_int_t matrix_layout, char compq,
                                char compz, armpl_int_t n, armpl_int_t ilo,
                                armpl_int_t ihi, float *a, armpl_int_t lda,
                                float *b, armpl_int_t ldb, float *q,
                                armpl_int_t ldq, float *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgghrd_work*, *LAPACKE_dgghrd_work*, *LAPACKE_sgghrd_work* and *LAPACKE_zgghrd_work*.

4.19.1371 LAPACKE_sgglse

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgglse(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t p, float *a,
                           armpl_int_t lda, float *b, armpl_int_t ldb,
                           float *c, float *d, float *x);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgglse*, *LAPACKE_dgglse*, *LAPACKE_sgglse* and *LAPACKE_zgglse*. It also exists with a native Fortran interface as *sgglse*.

4.19.1372 LAPACKE_sgglse_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgglse_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t p, float *a,
                                armpl_int_t lda, float *b, armpl_int_t ldb,
                                float *c, float *d, float *x, float *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgglse_work*, *LAPACKE_dgglse_work*, *LAPACKE_sgglse_work* and *LAPACKE_zgglse_work*.

4.19.1373 LAPACKE_sggqrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sggqrf(armpl_int_t matrix_layout, armpl_int_t n,
                           armpl_int_t m, armpl_int_t p, float *a,
                           armpl_int_t lda, float *taua, float *b,
                           armpl_int_t ldb, float *taub);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggqrf*, *LAPACKE_dggqrf*, *LAPACKE_sggqrf* and *LAPACKE_zggqrf*. It also exists with a native Fortran interface as *sggqrf*.

4.19.1374 LAPACKE_sggqrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sggqrf_work(armpl_int_t matrix_layout, armpl_int_t n,
                                armpl_int_t m, armpl_int_t p, float *a,
                                armpl_int_t lda, float *taua, float *b,
                                armpl_int_t ldb, float *taub, float *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggqrf_work*, *LAPACKE_dggqrf_work*, *LAPACKE_sggqrf_work* and *LAPACKE_zggqrf_work*.

4.19.1375 LAPACKE_sggrqf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sggrqf(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t p, armpl_int_t n, float *a,
                           armpl_int_t lda, float *taua, float *b,
                           armpl_int_t ldb, float *taub);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggrqf*, *LAPACKE_dggrqf*, *LAPACKE_sggrqf* and *LAPACKE_zggrqf*. It also exists with a native Fortran interface as *sggrqf*.

4.19.1376 LAPACKE_sggrqf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sggrqf_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t p, armpl_int_t n, float *a,
                                armpl_int_t lda, float *taua, float *b,
                                armpl_int_t ldb, float *taub, float *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggrqf_work*, *LAPACKE_dggrqf_work*, *LAPACKE_sggrqf_work* and *LAPACKE_zggrqf_work*.

4.19.1377 LAPACKE_sggsvd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sggsvd(armpl_int_t matrix_layout, char jobu, char jobv,
                           char jobq, armpl_int_t m, armpl_int_t n,
                           armpl_int_t p, armpl_int_t *k, armpl_int_t *l,
                           float *a, armpl_int_t lda, float *b,
                           armpl_int_t ldb, float *alpha, float *beta,
                           float *u, armpl_int_t ldu, float *v,
                           armpl_int_t ldv, float *q, armpl_int_t ldq,
                           armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggsvd*, *LAPACKE_dggsvd*, *LAPACKE_sggsvd* and *LAPACKE_zggsvd*.

4.19.1378 LAPACKE_sggsvd3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sggsvd3(armpl_int_t matrix_layout, char jobu, char jobv,
                            char jobq, armpl_int_t m, armpl_int_t n,
                            armpl_int_t p, armpl_int_t *k, armpl_int_t *l,
                            float *a, armpl_int_t lda, float *b,
                            armpl_int_t ldb, float *alpha, float *beta,
                            float *u, armpl_int_t ldu, float *v,
                            armpl_int_t ldv, float *q, armpl_int_t ldq,
                            armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggsvd3*, *LAPACKE_dggsvd3*, *LAPACKE_sggsvd3* and *LAPACKE_zggsvd3*. It also exists with a native Fortran interface as *sggsvd3*.

4.19.1379 LAPACKE_sggsvd3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sggsvd3_work(armpl_int_t matrix_layout, char jobu,
                                 char jobv, char jobq, armpl_int_t m,
                                 armpl_int_t n, armpl_int_t p, armpl_int_t *k,
                                 armpl_int_t *l, float *a, armpl_int_t lda,
                                 float *b, armpl_int_t ldb, float *alpha,
                                 float *beta, float *u, armpl_int_t ldu,
                                 float *v, armpl_int_t ldv, float *q,
                                 armpl_int_t ldq, float *work,
                                 armpl_int_t lwork, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggsvd3_work*, *LAPACKE_dggsvd3_work*, *LAPACKE_sggsvd3_work* and *LAPACKE_zggsvd3_work*.

4.19.1380 LAPACKE_sggsvd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sggsvd_work(armpl_int_t matrix_layout, char jobu,
                                char jobv, char jobq, armpl_int_t m,
                                armpl_int_t n, armpl_int_t p, armpl_int_t *k,
                                armpl_int_t *l, float *a, armpl_int_t lda,
                                float *b, armpl_int_t ldb, float *alpha,
                                float *beta, float *u, armpl_int_t ldu,
                                float *v, armpl_int_t ldv, float *q,
                                armpl_int_t ldq, float *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggsvd_work*, *LAPACKE_dggsvd_work*, *LAPACKE_sggsvd_work* and *LAPACKE_zggsvd_work*.

4.19.1381 LAPACKE_sggsvp

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sggsvp(armpl_int_t matrix_layout, char jobu, char jobv,
                           char jobq, armpl_int_t m, armpl_int_t p,
                           armpl_int_t n, float *a, armpl_int_t lda, float *b,
                           armpl_int_t ldb, float tola, float tolb,
                           armpl_int_t *k, armpl_int_t *l, float *u,
                           armpl_int_t ldu, float *v, armpl_int_t ldv,
                           float *q, armpl_int_t ldq);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggsvp*, *LAPACKE_dggsvp*, *LAPACKE_sggsvp* and *LAPACKE_zggsvp*.

4.19.1382 LAPACKE_sggsvp3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sggsvp3(armpl_int_t matrix_layout, char jobu, char jobv,
                            char jobq, armpl_int_t m, armpl_int_t p,
                            armpl_int_t n, float *a, armpl_int_t lda,
                            float *b, armpl_int_t ldb, float tola, float tolb,
                            armpl_int_t *k, armpl_int_t *l, float *u,
                            armpl_int_t ldu, float *v, armpl_int_t ldv,
                            float *q, armpl_int_t ldq);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggsvp3*, *LAPACKE_dggsvp3*, *LAPACKE_sggsvp3* and *LAPACKE_zggsvp3*. It also exists with a native Fortran interface as *sggsvp3*.

4.19.1383 LAPACKE_sggsvp3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sggsvp3_work(armpl_int_t matrix_layout, char jobu,
                                 char jobv, char jobq, armpl_int_t m,
                                 armpl_int_t p, armpl_int_t n, float *a,
                                 armpl_int_t lda, float *b, armpl_int_t ldb,
                                 float tola, float tolb, armpl_int_t *k,
                                 armpl_int_t *l, float *u, armpl_int_t ldu,
                                 float *v, armpl_int_t ldv, float *q,
                                 armpl_int_t ldq, armpl_int_t *iwork,
                                 float *tau, float *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggsvp3_work*, *LAPACKE_dggsvp3_work*, *LAPACKE_sggsvp3_work* and *LAPACKE_zggsvp3_work*.

4.19.1384 LAPACKE_sggsvp_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sggsvp_work(armpl_int_t matrix_layout, char jobu,
                                char jobv, char jobq, armpl_int_t m,
                                armpl_int_t p, armpl_int_t n, float *a,
                                armpl_int_t lda, float *b, armpl_int_t ldb,
                                float tola, float tolb, armpl_int_t *k,
                                armpl_int_t *l, float *u, armpl_int_t ldu,
                                float *v, armpl_int_t ldv, float *q,
                                armpl_int_t ldq, armpl_int_t *iwork,
                                float *tau, float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggsvp_work*, *LAPACKE_dggsvp_work*, *LAPACKE_sggsvp_work* and *LAPACKE_zggsvp_work*.

4.19.1385 LAPACKE_sgtcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgtcon(char norm, armpl_int_t n, const float *dl,
                           const float *d, const float *du, const float *du2,
                           const armpl_int_t *ipiv, float anorm,
                           float *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgtcon*, *LAPACKE_dgtcon*, *LAPACKE_sgtcon* and *LAPACKE_zgtcon*. It also exists with a native Fortran interface as *sgtcon*.

4.19.1386 LAPACKE_sgtcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgtcon_work(char norm, armpl_int_t n, const float *dl,
                                const float *d, const float *du,
                                const float *du2, const armpl_int_t *ipiv,
                                float anorm, float *rcond, float *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.1387 LAPACKE_sgtrfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgtrfs(armpl_int_t matrix_layout, char trans,
                           armpl_int_t n, armpl_int_t nrhs, const float *dl,
                           const float *d, const float *du, const float *dlf,
                           const float *df, const float *duf,
                           const float *du2, const armpl_int_t *ipiv,
                           const float *b, armpl_int_t ldb, float *x,
                           armpl_int_t ldx, float *ferr, float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgtrfs*, *LAPACKE_dgtrfs*, *LAPACKE_sgtrfs* and *LAPACKE_zgtrfs*. It also exists with a native Fortran interface as *sgtrfs*.

4.19.1388 LAPACKE_sgtrfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgtrfs_work(armpl_int_t matrix_layout, char trans,
                                armpl_int_t n, armpl_int_t nrhs,
                                const float *dl, const float *d,
                                const float *du, const float *dlf,
                                const float *df, const float *duf,
                                const float *du2, const armpl_int_t *ipiv,
                                const float *b, armpl_int_t ldb, float *x,
                                armpl_int_t ldx, float *ferr, float *berr,
                                float *work, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgtrfs_work*, *LAPACKE_dgtrfs_work*, *LAPACKE_sgtrfs_work* and *LAPACKE_zgtrfs_work*.

4.19.1389 LAPACKE_sgtsv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgtsv(armpl_int_t matrix_layout, armpl_int_t n,
                          armpl_int_t nrhs, float *dl, float *d, float *du,
                          float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgtsv*, *LAPACKE_dgtsv*, *LAPACKE_sgtsv* and *LAPACKE_zgtsv*. It also exists with a native Fortran interface as *sgtsv*.

4.19.1390 LAPACKE_sgtsv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgtsv_work(armpl_int_t matrix_layout, armpl_int_t n,
                               armpl_int_t nrhs, float *dl, float *d,
                               float *du, float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgtsv_work*, *LAPACKE_dgtsv_work*, *LAPACKE_sgtsv_work* and *LAPACKE_zgtsv_work*.

4.19.1391 LAPACKE_sgtsvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgtsvx(armpl_int_t matrix_layout, char fact, char trans,
                           armpl_int_t n, armpl_int_t nrhs, const float *dl,
                           const float *d, const float *du, float *dlf,
                           float *df, float *duf, float *du2,
                           armpl_int_t *ipiv, const float *b, armpl_int_t ldb,
                           float *x, armpl_int_t ldx, float *rcond,
                           float *ferr, float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgtsvx*, *LAPACKE_dgtsvx*, *LAPACKE_sgtsvx* and *LAPACKE_zgtsvx*. It also exists with a native Fortran interface as *sgtsvx*.

4.19.1392 LAPACKE_sgtsvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgtsvx_work(armpl_int_t matrix_layout, char fact,
                                char trans, armpl_int_t n, armpl_int_t nrhs,
```

(continues on next page)

(continued from previous page)

```
                                const float *dl, const float *d,
                                const float *du, float *dlf, float *df,
                                float *duf, float *du2, armpl_int_t *ipiv,
                                const float *b, armpl_int_t ldb, float *x,
                                armpl_int_t ldx, float *rcond, float *ferr,
                                float *berr, float *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgtsvx_work*, *LAPACKE_dgtsvx_work*, *LAPACKE_sgtsvx_work* and *LAPACKE_zgtsvx_work*.

4.19.1393 LAPACKE_sgttrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgttrf(armpl_int_t n, float *dl, float *d, float *du,
                           float *du2, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgttrf*, *LAPACKE_dgttrf*, *LAPACKE_sgttrf* and *LAPACKE_zgttrf*. It also exists with a native Fortran interface as *sgttrf*.

4.19.1394 LAPACKE_sgttrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgttrf_work(armpl_int_t n, float *dl, float *d, float *du,
                                float *du2, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgttrf_work*, *LAPACKE_dgttrf_work*, *LAPACKE_sgttrf_work* and *LAPACKE_zgttrf_work*.

4.19.1395 LAPACKE_sgttrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgttrs(armpl_int_t matrix_layout, char trans,
                           armpl_int_t n, armpl_int_t nrhs, const float *dl,
                           const float *d, const float *du, const float *du2,
                           const armpl_int_t *ipiv, float *b,
                           armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgttrs*, *LAPACKE_dgttrs*, *LAPACKE_sgttrs* and *LAPACKE_zgttrs*. It also exists with a native Fortran interface as *sgttrs*.

4.19.1396 LAPACKE_sgttrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sgttrs_work(armpl_int_t matrix_layout, char trans,
                                armpl_int_t n, armpl_int_t nrhs,
                                const float *dl, const float *d,
                                const float *du, const float *du2,
                                const armpl_int_t *ipiv, float *b,
                                armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgttrs_work*, *LAPACKE_dgttrs_work*, *LAPACKE_sgttrs_work* and *LAPACKE_zgttrs_work*.

4.19.1397 LAPACKE_shgeqz

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_shgeqz(armpl_int_t matrix_layout, char job, char compq,
                           char compz, armpl_int_t n, armpl_int_t ilo,
                           armpl_int_t ihi, float *h, armpl_int_t ldh,
                           float *t, armpl_int_t ldt, float *alphar,
                           float *alphai, float *beta, float *q,
                           armpl_int_t ldq, float *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chgeqz*, *LAPACKE_dhgeqz*, *LAPACKE_shgeqz* and *LAPACKE_zhgeqz*. It also exists with a native Fortran interface as *shgeqz*.

4.19.1398 LAPACKE_shgeqz_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_shgeqz_work(armpl_int_t matrix_layout, char job,
                                char compq, char compz, armpl_int_t n,
                                armpl_int_t ilo, armpl_int_t ihi, float *h,
                                armpl_int_t ldh, float *t, armpl_int_t ldt,
                                float *alphar, float *alphai, float *beta,
                                float *q, armpl_int_t ldq, float *z,
                                armpl_int_t ldz, float *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chgeqz_work*, *LAPACKE_dhgeqz_work*, *LAPACKE_shgeqz_work* and *LAPACKE_zhgeqz_work*.

4.19.1399 LAPACKE_shsein

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_shsein(armpl_int_t matrix_layout, char job, char eigsrc,
                           char initv, armpl_int_t *select, armpl_int_t n,
                           const float *h, armpl_int_t ldh, float *wr,
                           const float *wi, float *vl, armpl_int_t ldvl,
                           float *vr, armpl_int_t ldvr, armpl_int_t mm,
                           armpl_int_t *m, armpl_int_t *ifaill,
                           armpl_int_t *ifailr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chsein*, *LAPACKE_dhsein*, *LAPACKE_shsein* and *LAPACKE_zhsein*. It also exists with a native Fortran interface as *shsein*.

4.19.1400 LAPACKE_shsein_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_shsein_work(armpl_int_t matrix_layout, char job,
                                char eigsrc, char initv, armpl_int_t *select,
                                armpl_int_t n, const float *h,
                                armpl_int_t ldh, float *wr, const float *wi,
                                float *vl, armpl_int_t ldvl, float *vr,
                                armpl_int_t ldvr, armpl_int_t mm,
                                armpl_int_t *m, float *work,
                                armpl_int_t *ifaill, armpl_int_t *ifailr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chsein_work*, *LAPACKE_dhsein_work*, *LAPACKE_shsein_work* and *LAPACKE_zhsein_work*.

4.19.1401 LAPACKE_shseqr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_shseqr(armpl_int_t matrix_layout, char job, char compz,
                           armpl_int_t n, armpl_int_t ilo, armpl_int_t ihi,
                           float *h, armpl_int_t ldh, float *wr, float *wi,
                           float *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chseqr*, *LAPACKE_dhseqr*, *LAPACKE_shseqr* and *LAPACKE_zhseqr*. It also exists with a native Fortran interface as *shseqr*.

4.19.1402 LAPACKE_shseqr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_shseqr_work(armpl_int_t matrix_layout, char job,
                                char compz, armpl_int_t n, armpl_int_t ilo,
                                armpl_int_t ihi, float *h, armpl_int_t ldh,
                                float *wr, float *wi, float *z,
                                armpl_int_t ldz, float *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chseqr_work*, *LAPACKE_dhseqr_work*, *LAPACKE_shseqr_work* and *LAPACKE_zhseqr_work*.

4.19.1403 LAPACKE_slacn2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_slacn2(armpl_int_t n, float *v, float *x,
                           armpl_int_t *isgn, float *est, armpl_int_t *kase,
                           armpl_int_t *isave);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clacn2*, *LAPACKE_dlacn2*, *LAPACKE_slacn2* and *LAPACKE_zlacn2*. It also exists with a native Fortran interface as *slacn2*.

4.19.1404 LAPACKE_slacn2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_slacn2_work(armpl_int_t n, float *v, float *x,
                                armpl_int_t *isgn, float *est,
                                armpl_int_t *kase, armpl_int_t *isave);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clacn2_work*, *LAPACKE_dlacn2_work*, *LAPACKE_slacn2_work* and *LAPACKE_zlacn2_work*.

4.19.1405 LAPACKE_slacpy

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_slacpy(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t m, armpl_int_t n, const float *a,
                           armpl_int_t lda, float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clacpy*, *LAPACKE_dlacpy*, *LAPACKE_slacpy* and *LAPACKE_zlacpy*. It also exists with a native Fortran interface as *slacpy*.

4.19.1406 LAPACKE_slacpy_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_slacpy_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t m, armpl_int_t n, const float *a,
                                armpl_int_t lda, float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clacpy_work*, *LAPACKE_dlacpy_work*, *LAPACKE_slacpy_work* and *LAPACKE_zlacpy_work*.

4.19.1407 LAPACKE_slag2d

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_slag2d(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, const float *sa, armpl_int_t ldsa,
                           double *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_slag2d*. It also exists with a native Fortran interface as *slag2d*.

4.19.1408 LAPACKE_slag2d_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_slag2d_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, const float *sa,
                                armpl_int_t ldsa, double *a,
                                armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_slag2d_work*.

4.19.1409 LAPACKE_slagge

Syntax

Routine is deprecated. Please choose a suitable alternative.

4.19.1410 LAPACKE_slagge_work

Syntax

Routine is deprecated. Please choose a suitable alternative.

4.19.1411 LAPACKE_slagsy

Syntax

Routine is deprecated. Please choose a suitable alternative.

4.19.1412 LAPACKE_slagsy_work

Syntax

Routine is deprecated. Please choose a suitable alternative.

4.19.1413 LAPACKE_slamch

Syntax

```
#include "armpl.h"

float LAPACKE_slamch(char cmach);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dlamch* and *LAPACKE_slamch*.

4.19.1414 LAPACKE_slamch_work

Syntax

```
#include "armpl.h"

float LAPACKE_slamch_work(char cmach);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dlamch_work* and *LAPACKE_slamch_work*.

4.19.1415 LAPACKE_slange

Syntax

```
#include "armpl.h"

float LAPACKE_slange(armpl_int_t matrix_layout, char norm, armpl_int_t m,
                     armpl_int_t n, const float *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clange*, *LAPACKE_dlange*, *LAPACKE_slange* and *LAPACKE_zlange*. It also exists with a native Fortran interface as *slange*.

4.19.1416 LAPACKE_slange_work

Syntax

```
#include "armpl.h"

float LAPACKE_slange_work(armpl_int_t matrix_layout, char norm, armpl_int_t m,
                          armpl_int_t n, const float *a, armpl_int_t lda,
                          float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clange_work*, *LAPACKE_dlange_work*, *LAPACKE_slange_work* and *LAPACKE_zlange_work*.

4.19.1417 LAPACKE_slansy

Syntax

```
#include "armpl.h"

float LAPACKE_slansy(armpl_int_t matrix_layout, char norm, char uplo,
                     armpl_int_t n, const float *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clansy*, *LAPACKE_dlansy*, *LAPACKE_slansy* and *LAPACKE_zlansy*. It also exists with a native Fortran interface as *slansy*.

4.19.1418 LAPACKE_slansy_work

Syntax

```
#include "armpl.h"

float LAPACKE_slansy_work(armpl_int_t matrix_layout, char norm, char uplo,
                          armpl_int_t n, const float *a, armpl_int_t lda,
                          float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clansy_work*, *LAPACKE_dlansy_work*, *LAPACKE_slansy_work* and *LAPACKE_zlansy_work*.

4.19.1419 LAPACKE_slantr

Syntax

```
#include "armpl.h"

float LAPACKE_slantr(armpl_int_t matrix_layout, char norm, char uplo,
                     char diag, armpl_int_t m, armpl_int_t n, const float *a,
                     armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clantr*, *LAPACKE_dlantr*, *LAPACKE_slantr* and *LAPACKE_zlantr*. It also exists with a native Fortran interface as *slantr*.

4.19.1420 LAPACKE_slantr_work

Syntax

```
#include "armpl.h"

float LAPACKE_slantr_work(armpl_int_t matrix_layout, char norm, char uplo,
                          char diag, armpl_int_t m, armpl_int_t n,
                          const float *a, armpl_int_t lda, float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clantr_work*, *LAPACKE_dlantr_work*, *LAPACKE_slantr_work* and *LAPACKE_zlantr_work*.

4.19.1421 LAPACKE_slapmr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_slapmr(armpl_int_t matrix_layout, armpl_int_t forwrd,
                           armpl_int_t m, armpl_int_t n, float *x,
                           armpl_int_t ldx, armpl_int_t *k);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clapmr*, *LAPACKE_dlapmr*, *LAPACKE_slapmr* and *LAPACKE_zlapmr*. It also exists with a native Fortran interface as *slapmr*.

4.19.1422 LAPACKE_slapmr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_slapmr_work(armpl_int_t matrix_layout, armpl_int_t forwrd,
                                armpl_int_t m, armpl_int_t n, float *x,
                                armpl_int_t ldx, armpl_int_t *k);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clapmr_work*, *LAPACKE_dlapmr_work*, *LAPACKE_slapmr_work* and *LAPACKE_zlapmr_work*.

4.19.1423 LAPACKE_slapmt

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_slapmt(armpl_int_t matrix_layout, armpl_int_t forwrd,
                           armpl_int_t m, armpl_int_t n, float *x,
                           armpl_int_t ldx, armpl_int_t *k);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clapmt*, *LAPACKE_dlapmt*, *LAPACKE_slapmt* and *LAPACKE_zlapmt*. It also exists with a native Fortran interface as *slapmt*.

4.19.1424 LAPACKE_slapmt_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_slapmt_work(armpl_int_t matrix_layout, armpl_int_t forwrd,
                                armpl_int_t m, armpl_int_t n, float *x,
                                armpl_int_t ldx, armpl_int_t *k);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.1425 LAPACKE_slapy2

Syntax

```
#include "armpl.h"

float LAPACKE_slapy2(float x, float y);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dlapy2* and *LAPACKE_slapy2*. It also exists with a native Fortran interface as *slapy2*.

4.19.1426 LAPACKE_slapy2_work

Syntax

```
#include "armpl.h"

float LAPACKE_slapy2_work(float x, float y);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dlapy2_work* and *LAPACKE_slapy2_work*.

4.19.1427 LAPACKE_slapy3

Syntax

```
#include "armpl.h"

float LAPACKE_slapy3(float x, float y, float z);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dlapy3* and *LAPACKE_slapy3*. It also exists with a native Fortran interface as *slapy3*.

4.19.1428 LAPACKE_slapy3_work

Syntax

```
#include "armpl.h"

float LAPACKE_slapy3_work(float x, float y, float z);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dlapy3_work* and *LAPACKE_slapy3_work*.

4.19.1429 LAPACKE_slarfb

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_slarfb(armpl_int_t matrix_layout, char side, char trans,
                           char direct, char storev, armpl_int_t m,
                           armpl_int_t n, armpl_int_t k, const float *v,
                           armpl_int_t ldv, const float *t, armpl_int_t ldt,
                           float *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarfb*, *LAPACKE_dlarfb*, *LAPACKE_slarfb* and *LAPACKE_zlarfb*. It also exists with a native Fortran interface as *slarfb*.

4.19.1430 LAPACKE_slarfb_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_slarfb_work(armpl_int_t matrix_layout, char side,
                                char trans, char direct, char storev,
                                armpl_int_t m, armpl_int_t n, armpl_int_t k,
                                const float *v, armpl_int_t ldv,
                                const float *t, armpl_int_t ldt, float *c,
                                armpl_int_t ldc, float *work,
                                armpl_int_t ldwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarfb_work*, *LAPACKE_dlarfb_work*, *LAPACKE_slarfb_work* and *LAPACKE_zlarfb_work*.

4.19.1431 LAPACKE_slarfg

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_slarfg(armpl_int_t n, float *alpha, float *x,
                           armpl_int_t incx, float *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarfg*, *LAPACKE_dlarfg*, *LAPACKE_slarfg* and *LAPACKE_zlarfg*. It also exists with a native Fortran interface as *slarfg*.

4.19.1432 LAPACKE_slarfg_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_slarfg_work(armpl_int_t n, float *alpha, float *x,
                                armpl_int_t incx, float *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarfg_work*, *LAPACKE_dlarfg_work*, *LAPACKE_slarfg_work* and *LAPACKE_zlarfg_work*.

4.19.1433 LAPACKE_slarft

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_slarft(armpl_int_t matrix_layout, char direct,
                           char storev, armpl_int_t n, armpl_int_t k,
                           const float *v, armpl_int_t ldv, const float *tau,
                           float *t, armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarft*, *LAPACKE_dlarft*, *LAPACKE_slarft* and *LAPACKE_zlarft*. It also exists with a native Fortran interface as *slarft*.

4.19.1434 LAPACKE_slarft_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_slarft_work(armpl_int_t matrix_layout, char direct,
                                char storev, armpl_int_t n, armpl_int_t k,
                                const float *v, armpl_int_t ldv,
                                const float *tau, float *t, armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarft_work*, *LAPACKE_dlarft_work*, *LAPACKE_slarft_work* and *LAPACKE_zlarft_work*.

4.19.1435 LAPACKE_slarfx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_slarfx(armpl_int_t matrix_layout, char side,
                           armpl_int_t m, armpl_int_t n, const float *v,
                           float tau, float *c, armpl_int_t ldc,
                           float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarfx*, *LAPACKE_dlarfx*, *LAPACKE_slarfx* and *LAPACKE_zlarfx*. It also exists with a native Fortran interface as *slarfx*.

4.19.1436 LAPACKE_slarfx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_slarfx_work(armpl_int_t matrix_layout, char side,
                                armpl_int_t m, armpl_int_t n, const float *v,
                                float tau, float *c, armpl_int_t ldc,
                                float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarfx_work*, *LAPACKE_dlarfx_work*, *LAPACKE_slarfx_work* and *LAPACKE_zlarfx_work*.

4.19.1437 LAPACKE_slarnv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_slarnv(armpl_int_t idist, armpl_int_t *iseed,
                           armpl_int_t n, float *x);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarnv*, *LAPACKE_dlarnv*, *LAPACKE_slarnv* and *LAPACKE_zlarnv*. It also exists with a native Fortran interface as *slarnv*.

4.19.1438 LAPACKE_slarnv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_slarnv_work(armpl_int_t idist, armpl_int_t *iseed,
                                armpl_int_t n, float *x);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarnv_work*, *LAPACKE_dlarnv_work*, *LAPACKE_slarnv_work* and *LAPACKE_zlarnv_work*.

4.19.1439 LAPACKE_slartgp

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_slartgp(float f, float g, float *cs, float *sn,
                            float *r);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dlartgp* and *LAPACKE_slartgp*. It also exists with a native Fortran interface as *slartgp*.

4.19.1440 LAPACKE_slartgp_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_slartgp_work(float f, float g, float *cs, float *sn,
                                 float *r);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dlartgp_work* and *LAPACKE_slartgp_work*.

4.19.1441 LAPACKE_slartgs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_slartgs(float x, float y, float sigma, float *cs,
                            float *sn);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dlartgs* and *LAPACKE_slartgs*. It also exists with a native Fortran interface as *slartgs*.

4.19.1442 LAPACKE_slartgs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_slartgs_work(float x, float y, float sigma, float *cs,
                                 float *sn);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dlartgs_work* and *LAPACKE_slartgs_work*.

4.19.1443 LAPACKE_slascl

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_slascl(armpl_int_t matrix_layout, char type,
                           armpl_int_t kl, armpl_int_t ku, float cfrom,
                           float cto, armpl_int_t m, armpl_int_t n, float *a,
                           armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.1444 LAPACKE_slascl_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_slascl_work(armpl_int_t matrix_layout, char type,
                                armpl_int_t kl, armpl_int_t ku, float cfrom,
                                float cto, armpl_int_t m, armpl_int_t n,
                                float *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clascl_work*, *LAPACKE_dlascl_work*, *LAPACKE_slascl_work* and *LAPACKE_zlascl_work*.

4.19.1445 LAPACKE_slaset

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_slaset(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t m, armpl_int_t n, float alpha,
                           float beta, float *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_claset*, *LAPACKE_dlaset*, *LAPACKE_slaset* and *LAPACKE_zlaset*. It also exists with a native Fortran interface as *slaset*.

4.19.1446 LAPACKE_slaset_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_slaset_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t m, armpl_int_t n, float alpha,
                                float beta, float *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_claset_work*, *LAPACKE_dlaset_work*, *LAPACKE_slaset_work* and *LAPACKE_zlaset_work*.

4.19.1447 LAPACKE_slasrt

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_slasrt(char id, armpl_int_t n, float *d);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dlasrt* and *LAPACKE_slasrt*. It also exists with a native Fortran interface as *slasrt*.

4.19.1448 LAPACKE_slasrt_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_slasrt_work(char id, armpl_int_t n, float *d);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dlasrt_work* and *LAPACKE_slasrt_work*.

4.19.1449 LAPACKE_slassq

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_slassq(armpl_int_t n, float *x, armpl_int_t incx,
                           float *scale, float *sumsq);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_classq*, *LAPACKE_dlassq*, *LAPACKE_slassq* and *LAPACKE_zlassq*. It also exists with a native Fortran interface as *slassq*.

4.19.1450 LAPACKE_slassq_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_slassq_work(armpl_int_t n, float *x, armpl_int_t incx,
                                float *scale, float *sumsq);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_classq_work*, *LAPACKE_dlassq_work*, *LAPACKE_slassq_work* and *LAPACKE_zlassq_work*.

4.19.1451 LAPACKE_slaswp

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_slaswp(armpl_int_t matrix_layout, armpl_int_t n, float *a,
                           armpl_int_t lda, armpl_int_t k1, armpl_int_t k2,
                           const armpl_int_t *ipiv, armpl_int_t incx);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_claswp*, *LAPACKE_dlaswp*, *LAPACKE_slaswp* and *LAPACKE_zlaswp*. It also exists with a native Fortran interface as *slaswp*.

4.19.1452 LAPACKE_slaswp_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_slaswp_work(armpl_int_t matrix_layout, armpl_int_t n,
                                float *a, armpl_int_t lda, armpl_int_t k1,
                                armpl_int_t k2, const armpl_int_t *ipiv,
                                armpl_int_t incx);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_claswp_work*, *LAPACKE_dlaswp_work*, *LAPACKE_slaswp_work* and *LAPACKE_zlaswp_work*.

4.19.1453 LAPACKE_slatms

Syntax

Routine is deprecated. Please choose a suitable alternative.

4.19.1454 LAPACKE_slatms_work

Syntax

Routine is deprecated. Please choose a suitable alternative.

4.19.1455 LAPACKE_slauum

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_slauum(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, float *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clauum*, *LAPACKE_dlauum*, *LAPACKE_slauum* and *LAPACKE_zlauum*. It also exists with a native Fortran interface as *slauum*.

4.19.1456 LAPACKE_slauum_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_slauum_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, float *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clauum_work*, *LAPACKE_dlauum_work*, *LAPACKE_slauum_work* and *LAPACKE_zlauum_work*.

4.19.1457 LAPACKE_sopgtr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sopgtr(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, const float *ap, const float *tau,
                           float *q, armpl_int_t ldq);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dopgtr* and *LAPACKE_sopgtr*. It also exists with a native Fortran interface as *sopgtr*.

4.19.1458 LAPACKE_sopgtr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sopgtr_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, const float *ap,
                                const float *tau, float *q, armpl_int_t ldq,
                                float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dopgtr_work* and *LAPACKE_sopgtr_work*.

4.19.1459 LAPACKE_sopmtr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sopmtr(armpl_int_t matrix_layout, char side, char uplo,
                           char trans, armpl_int_t m, armpl_int_t n,
                           const float *ap, const float *tau, float *c,
                           armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dopmtr* and *LAPACKE_sopmtr*. It also exists with a native Fortran interface as *sopmtr*.

4.19.1460 LAPACKE_sopmtr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sopmtr_work(armpl_int_t matrix_layout, char side,
                                char uplo, char trans, armpl_int_t m,
                                armpl_int_t n, const float *ap,
                                const float *tau, float *c, armpl_int_t ldc,
                                float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dopmtr_work* and *LAPACKE_sopmtr_work*.

4.19.1461 LAPACKE_sorbdb

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sorbdb(armpl_int_t matrix_layout, char trans, char signs,
                           armpl_int_t m, armpl_int_t p, armpl_int_t q,
                           float *x11, armpl_int_t ldx11, float *x12,
                           armpl_int_t ldx12, float *x21, armpl_int_t ldx21,
                           float *x22, armpl_int_t ldx22, float *theta,
                           float *phi, float *taup1, float *taup2,
                           float *tauq1, float *tauq2);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorbdb* and *LAPACKE_sorbdb*. It also exists with a native Fortran interface as *sorbdb*.

4.19.1462 LAPACKE_sorbdb_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sorbdb_work(armpl_int_t matrix_layout, char trans,
                                char signs, armpl_int_t m, armpl_int_t p,
                                armpl_int_t q, float *x11, armpl_int_t ldx11,
                                float *x12, armpl_int_t ldx12, float *x21,
                                armpl_int_t ldx21, float *x22,
                                armpl_int_t ldx22, float *theta, float *phi,
                                float *taup1, float *taup2, float *tauq1,
                                float *tauq2, float *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorbdb_work* and *LAPACKE_sorbdb_work*.

4.19.1463 LAPACKE_sorcsd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sorcsd(armpl_int_t matrix_layout, char jobu1, char jobu2,
                           char jobv1t, char jobv2t, char trans, char signs,
                           armpl_int_t m, armpl_int_t p, armpl_int_t q,
                           float *x11, armpl_int_t ldx11, float *x12,
                           armpl_int_t ldx12, float *x21, armpl_int_t ldx21,
                           float *x22, armpl_int_t ldx22, float *theta,
                           float *u1, armpl_int_t ldu1, float *u2,
                           armpl_int_t ldu2, float *v1t, armpl_int_t ldv1t,
                           float *v2t, armpl_int_t ldv2t);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorcsd* and *LAPACKE_sorcsd*. It also exists with a native Fortran interface as *sorcsd*.

4.19.1464 LAPACKE_sorcsd2by1

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sorcsd2by1(armpl_int_t matrix_layout, char jobu1,
                               char jobu2, char jobv1t, armpl_int_t m,
                               armpl_int_t p, armpl_int_t q, float *x11,
                               armpl_int_t ldx11, float *x21,
                               armpl_int_t ldx21, float *theta, float *u1,
                               armpl_int_t ldu1, float *u2, armpl_int_t ldu2,
                               float *v1t, armpl_int_t ldv1t);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorcsd2by1* and *LAPACKE_sorcsd2by1*. It also exists with a native Fortran interface as *sorcsd2by1*.

4.19.1465 LAPACKE_sorcsd2by1_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sorcsd2by1_work(armpl_int_t matrix_layout, char jobu1,
                                    char jobu2, char jobv1t, armpl_int_t m,
                                    armpl_int_t p, armpl_int_t q, float *x11,
                                    armpl_int_t ldx11, float *x21,
                                    armpl_int_t ldx21, float *theta,
                                    float *u1, armpl_int_t ldu1, float *u2,
                                    armpl_int_t ldu2, float *v1t,
                                    armpl_int_t ldv1t, float *work,
                                    armpl_int_t lwork, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorcsd2by1_work* and *LAPACKE_sorcsd2by1_work*.

4.19.1466 LAPACKE_sorcsd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sorcsd_work(armpl_int_t matrix_layout, char jobu1,
                                char jobu2, char jobv1t, char jobv2t,
                                char trans, char signs, armpl_int_t m,
                                armpl_int_t p, armpl_int_t q, float *x11,
                                armpl_int_t ldx11, float *x12,
                                armpl_int_t ldx12, float *x21,
                                armpl_int_t ldx21, float *x22,
```

(continues on next page)

(continued from previous page)

```
                                armpl_int_t ldx22, float *theta, float *u1,
                                armpl_int_t ldu1, float *u2, armpl_int_t ldu2,
                                float *v1t, armpl_int_t ldv1t, float *v2t,
                                armpl_int_t ldv2t, float *work,
                                armpl_int_t lwork, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorcsd_work* and *LAPACKE_sorcsd_work*.

4.19.1467 LAPACKE_sorgbr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sorgbr(armpl_int_t matrix_layout, char vect,
                           armpl_int_t m, armpl_int_t n, armpl_int_t k,
                           float *a, armpl_int_t lda, const float *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorgbr* and *LAPACKE_sorgbr*. It also exists with a native Fortran interface as *sorgbr*.

4.19.1468 LAPACKE_sorgbr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sorgbr_work(armpl_int_t matrix_layout, char vect,
                                armpl_int_t m, armpl_int_t n, armpl_int_t k,
                                float *a, armpl_int_t lda, const float *tau,
                                float *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorgbr_work* and *LAPACKE_sorgbr_work*.

4.19.1469 LAPACKE_sorghr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sorghr(armpl_int_t matrix_layout, armpl_int_t n,
                           armpl_int_t ilo, armpl_int_t ihi, float *a,
                           armpl_int_t lda, const float *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorghr* and *LAPACKE_sorghr*. It also exists with a native Fortran interface as *sorghr*.

4.19.1470 LAPACKE_sorghr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sorghr_work(armpl_int_t matrix_layout, armpl_int_t n,
                                armpl_int_t ilo, armpl_int_t ihi, float *a,
                                armpl_int_t lda, const float *tau,
                                float *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorghr_work* and *LAPACKE_sorghr_work*.

4.19.1471 LAPACKE_sorglq

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sorglq(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t k, float *a,
                           armpl_int_t lda, const float *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorglq* and *LAPACKE_sorglq*. It also exists with a native Fortran interface as *sorglq*.

4.19.1472 LAPACKE_sorglq_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sorglq_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t k, float *a,
                                armpl_int_t lda, const float *tau,
                                float *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorglq_work* and *LAPACKE_sorglq_work*.

4.19.1473 LAPACKE_sorgql

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sorgql(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t k, float *a,
                           armpl_int_t lda, const float *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorgql* and *LAPACKE_sorgql*. It also exists with a native Fortran interface as *sorgql*.

4.19.1474 LAPACKE_sorgql_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sorgql_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t k, float *a,
                                armpl_int_t lda, const float *tau,
                                float *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorgql_work* and *LAPACKE_sorgql_work*.

4.19.1475 LAPACKE_sorgqr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sorgqr(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t k, float *a,
                           armpl_int_t lda, const float *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorgqr* and *LAPACKE_sorgqr*. It also exists with a native Fortran interface as *sorgqr*.

4.19.1476 LAPACKE_sorgqr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sorgqr_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t k, float *a,
                                armpl_int_t lda, const float *tau,
                                float *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorgqr_work* and *LAPACKE_sorgqr_work*.

4.19.1477 LAPACKE_sorgrq

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sorgrq(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t k, float *a,
                           armpl_int_t lda, const float *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorgrq* and *LAPACKE_sorgrq*. It also exists with a native Fortran interface as *sorgrq*.

4.19.1478 LAPACKE_sorgrq_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sorgrq_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t k, float *a,
                                armpl_int_t lda, const float *tau,
                                float *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorgrq_work* and *LAPACKE_sorgrq_work*.

4.19.1479 LAPACKE_sorgtr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sorgtr(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, float *a, armpl_int_t lda,
                           const float *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorgtr* and *LAPACKE_sorgtr*. It also exists with a native Fortran interface as *sorgtr*.

4.19.1480 LAPACKE_sorgtr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sorgtr_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, float *a, armpl_int_t lda,
                                const float *tau, float *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dorgtr_work* and *LAPACKE_sorgtr_work*.

4.19.1481 LAPACKE_sormbr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sormbr(armpl_int_t matrix_layout, char vect, char side,
                           char trans, armpl_int_t m, armpl_int_t n,
                           armpl_int_t k, const float *a, armpl_int_t lda,
                           const float *tau, float *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dormbr* and *LAPACKE_sormbr*. It also exists with a native Fortran interface as *sormbr*.

4.19.1482 LAPACKE_sormbr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sormbr_work(armpl_int_t matrix_layout, char vect,
                                char side, char trans, armpl_int_t m,
                                armpl_int_t n, armpl_int_t k, const float *a,
                                armpl_int_t lda, const float *tau, float *c,
                                armpl_int_t ldc, float *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dormbr_work* and *LAPACKE_sormbr_work*.

4.19.1483 LAPACKE_sormhr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sormhr(armpl_int_t matrix_layout, char side, char trans,
                           armpl_int_t m, armpl_int_t n, armpl_int_t ilo,
                           armpl_int_t ihi, const float *a, armpl_int_t lda,
                           const float *tau, float *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dormhr* and *LAPACKE_sormhr*. It also exists with a native Fortran interface as *sormhr*.

4.19.1484 LAPACKE_sormhr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sormhr_work(armpl_int_t matrix_layout, char side,
                                char trans, armpl_int_t m, armpl_int_t n,
                                armpl_int_t ilo, armpl_int_t ihi,
                                const float *a, armpl_int_t lda,
                                const float *tau, float *c, armpl_int_t ldc,
                                float *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dormhr_work* and *LAPACKE_sormhr_work*.

4.19.1485 LAPACKE_sormlq

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sormlq(armpl_int_t matrix_layout, char side, char trans,
                           armpl_int_t m, armpl_int_t n, armpl_int_t k,
                           const float *a, armpl_int_t lda, const float *tau,
                           float *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dormlq* and *LAPACKE_sormlq*. It also exists with a native Fortran interface as *sormlq*.

4.19.1486 LAPACKE_sormlq_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sormlq_work(armpl_int_t matrix_layout, char side,
                                char trans, armpl_int_t m, armpl_int_t n,
                                armpl_int_t k, const float *a,
                                armpl_int_t lda, const float *tau, float *c,
                                armpl_int_t ldc, float *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dormlq_work* and *LAPACKE_sormlq_work*.

4.19.1487 LAPACKE_sormql

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sormql(armpl_int_t matrix_layout, char side, char trans,
                           armpl_int_t m, armpl_int_t n, armpl_int_t k,
                           const float *a, armpl_int_t lda, const float *tau,
                           float *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dormql* and *LAPACKE_sormql*. It also exists with a native Fortran interface as *sormql*.

4.19.1488 LAPACKE_sormql_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sormql_work(armpl_int_t matrix_layout, char side,
                                char trans, armpl_int_t m, armpl_int_t n,
                                armpl_int_t k, const float *a,
                                armpl_int_t lda, const float *tau, float *c,
                                armpl_int_t ldc, float *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dormql_work* and *LAPACKE_sormql_work*.

4.19.1489 LAPACKE_sormqr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sormqr(armpl_int_t matrix_layout, char side, char trans,
                           armpl_int_t m, armpl_int_t n, armpl_int_t k,
                           const float *a, armpl_int_t lda, const float *tau,
                           float *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dormqr* and *LAPACKE_sormqr*. It also exists with a native Fortran interface as *sormqr*.

4.19.1490 LAPACKE_sormqr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sormqr_work(armpl_int_t matrix_layout, char side,
                                char trans, armpl_int_t m, armpl_int_t n,
                                armpl_int_t k, const float *a,
                                armpl_int_t lda, const float *tau, float *c,
                                armpl_int_t ldc, float *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dormqr_work* and *LAPACKE_sormqr_work*.

4.19.1491 LAPACKE_sormrq

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sormrq(armpl_int_t matrix_layout, char side, char trans,
                           armpl_int_t m, armpl_int_t n, armpl_int_t k,
                           const float *a, armpl_int_t lda, const float *tau,
                           float *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dormrq* and *LAPACKE_sormrq*. It also exists with a native Fortran interface as *sormrq*.

4.19.1492 LAPACKE_sormrq_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sormrq_work(armpl_int_t matrix_layout, char side,
                                char trans, armpl_int_t m, armpl_int_t n,
                                armpl_int_t k, const float *a,
                                armpl_int_t lda, const float *tau, float *c,
                                armpl_int_t ldc, float *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dormrq_work* and *LAPACKE_sormrq_work*.

4.19.1493 LAPACKE_sormrz

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sormrz(armpl_int_t matrix_layout, char side, char trans,
                           armpl_int_t m, armpl_int_t n, armpl_int_t k,
                           armpl_int_t l, const float *a, armpl_int_t lda,
                           const float *tau, float *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dormrz* and *LAPACKE_sormrz*. It also exists with a native Fortran interface as *sormrz*.

4.19.1494 LAPACKE_sormrz_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sormrz_work(armpl_int_t matrix_layout, char side,
                                char trans, armpl_int_t m, armpl_int_t n,
                                armpl_int_t k, armpl_int_t l, const float *a,
                                armpl_int_t lda, const float *tau, float *c,
                                armpl_int_t ldc, float *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dormrz_work* and *LAPACKE_sormrz_work*.

4.19.1495 LAPACKE_sormtr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sormtr(armpl_int_t matrix_layout, char side, char uplo,
                           char trans, armpl_int_t m, armpl_int_t n,
                           const float *a, armpl_int_t lda, const float *tau,
                           float *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dormtr* and *LAPACKE_sormtr*. It also exists with a native Fortran interface as *sormtr*.

4.19.1496 LAPACKE_sormtr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sormtr_work(armpl_int_t matrix_layout, char side,
                                char uplo, char trans, armpl_int_t m,
                                armpl_int_t n, const float *a,
                                armpl_int_t lda, const float *tau, float *c,
                                armpl_int_t ldc, float *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dormtr_work* and *LAPACKE_sormtr_work*.

4.19.1497 LAPACKE_spbcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spbcon(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t kd, const float *ab,
                           armpl_int_t ldab, float anorm, float *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbcon*, *LAPACKE_dpbcon*, *LAPACKE_spbcon* and *LAPACKE_zpbcon*. It also exists with a native Fortran interface as *spbcon*.

4.19.1498 LAPACKE_spbcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spbcon_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t kd,
                                const float *ab, armpl_int_t ldab,
                                float anorm, float *rcond, float *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbcon_work*, *LAPACKE_dpbcon_work*, *LAPACKE_spbcon_work* and *LAPACKE_zpbcon_work*.

4.19.1499 LAPACKE_spbequ

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spbequ(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t kd, const float *ab,
                           armpl_int_t ldab, float *s, float *scond,
                           float *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbequ*, *LAPACKE_dpbequ*, *LAPACKE_spbequ* and *LAPACKE_zpbequ*. It also exists with a native Fortran interface as *spbequ*.

4.19.1500 LAPACKE_spbequ_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spbequ_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t kd,
                                const float *ab, armpl_int_t ldab, float *s,
                                float *scond, float *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbequ_work*, *LAPACKE_dpbequ_work*, *LAPACKE_spbequ_work* and *LAPACKE_zpbequ_work*.

4.19.1501 LAPACKE_spbrfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spbrfs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t kd, armpl_int_t nrhs,
                           const float *ab, armpl_int_t ldab,
                           const float *afb, armpl_int_t ldafb,
                           const float *b, armpl_int_t ldb, float *x,
                           armpl_int_t ldx, float *ferr, float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbrfs*, *LAPACKE_dpbrfs*, *LAPACKE_spbrfs* and *LAPACKE_zpbrfs*. It also exists with a native Fortran interface as *spbrfs*.

4.19.1502 LAPACKE_spbrfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spbrfs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t kd,
                                armpl_int_t nrhs, const float *ab,
                                armpl_int_t ldab, const float *afb,
                                armpl_int_t ldafb, const float *b,
                                armpl_int_t ldb, float *x, armpl_int_t ldx,
                                float *ferr, float *berr, float *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbrfs_work*, *LAPACKE_dpbrfs_work*, *LAPACKE_spbrfs_work* and *LAPACKE_zpbrfs_work*.

4.19.1503 LAPACKE_spbstf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spbstf(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t kb, float *bb,
                           armpl_int_t ldbb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbstf*, *LAPACKE_dpbstf*, *LAPACKE_spbstf* and *LAPACKE_zpbstf*. It also exists with a native Fortran interface as *spbstf*.

4.19.1504 LAPACKE_spbstf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spbstf_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t kb, float *bb,
                                armpl_int_t ldbb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbstf_work*, *LAPACKE_dpbstf_work*, *LAPACKE_spbstf_work* and *LAPACKE_zpbstf_work*.

4.19.1505 LAPACKE_spbsv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spbsv(armpl_int_t matrix_layout, char uplo, armpl_int_t n,
                          armpl_int_t kd, armpl_int_t nrhs, float *ab,
                          armpl_int_t ldab, float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbsv*, *LAPACKE_dpbsv*, *LAPACKE_spbsv* and *LAPACKE_zpbsv*. It also exists with a native Fortran interface as *spbsv*.

4.19.1506 LAPACKE_spbsv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spbsv_work(armpl_int_t matrix_layout, char uplo,
                               armpl_int_t n, armpl_int_t kd,
                               armpl_int_t nrhs, float *ab, armpl_int_t ldab,
                               float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbsv_work*, *LAPACKE_dpbsv_work*, *LAPACKE_spbsv_work* and *LAPACKE_zpbsv_work*.

4.19.1507 LAPACKE_spbsvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spbsvx(armpl_int_t matrix_layout, char fact, char uplo,
                           armpl_int_t n, armpl_int_t kd, armpl_int_t nrhs,
                           float *ab, armpl_int_t ldab, float *afb,
                           armpl_int_t ldafb, char *equed, float *s, float *b,
                           armpl_int_t ldb, float *x, armpl_int_t ldx,
                           float *rcond, float *ferr, float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbsvx*, *LAPACKE_dpbsvx*, *LAPACKE_spbsvx* and *LAPACKE_zpbsvx*. It also exists with a native Fortran interface as *spbsvx*.

4.19.1508 LAPACKE_spbsvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spbsvx_work(armpl_int_t matrix_layout, char fact,
                                char uplo, armpl_int_t n, armpl_int_t kd,
                                armpl_int_t nrhs, float *ab, armpl_int_t ldab,
                                float *afb, armpl_int_t ldafb, char *equed,
                                float *s, float *b, armpl_int_t ldb, float *x,
                                armpl_int_t ldx, float *rcond, float *ferr,
                                float *berr, float *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbsvx_work*, *LAPACKE_dpbsvx_work*, *LAPACKE_spbsvx_work* and *LAPACKE_zpbsvx_work*.

4.19.1509 LAPACKE_spbtrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spbtrf(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t kd, float *ab,
                           armpl_int_t ldab);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbtrf*, *LAPACKE_dpbtrf*, *LAPACKE_spbtrf* and *LAPACKE_zpbtrf*. It also exists with a native Fortran interface as *spbtrf*.

4.19.1510 LAPACKE_spbtrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spbtrf_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t kd, float *ab,
                                armpl_int_t ldab);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbtrf_work*, *LAPACKE_dpbtrf_work*, *LAPACKE_spbtrf_work* and *LAPACKE_zpbtrf_work*.

4.19.1511 LAPACKE_spbtrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spbtrs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t kd, armpl_int_t nrhs,
                           const float *ab, armpl_int_t ldab, float *b,
                           armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbtrs*, *LAPACKE_dpbtrs*, *LAPACKE_spbtrs* and *LAPACKE_zpbtrs*. It also exists with a native Fortran interface as *spbtrs*.

4.19.1512 LAPACKE_spbtrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spbtrs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t kd,
                                armpl_int_t nrhs, const float *ab,
                                armpl_int_t ldab, float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbtrs_work*, *LAPACKE_dpbtrs_work*, *LAPACKE_spbtrs_work* and *LAPACKE_zpbtrs_work*.

4.19.1513 LAPACKE_spftrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spftrf(armpl_int_t matrix_layout, char transr, char uplo,
                           armpl_int_t n, float *a);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpftrf*, *LAPACKE_dpftrf*, *LAPACKE_spftrf* and *LAPACKE_zpftrf*. It also exists with a native Fortran interface as *spftrf*.

4.19.1514 LAPACKE_spftrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spftrf_work(armpl_int_t matrix_layout, char transr,
                                char uplo, armpl_int_t n, float *a);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpftrf_work*, *LAPACKE_dpftrf_work*, *LAPACKE_spftrf_work* and *LAPACKE_zpftrf_work*.

4.19.1515 LAPACKE_spftri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spftri(armpl_int_t matrix_layout, char transr, char uplo,
                           armpl_int_t n, float *a);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpftri*, *LAPACKE_dpftri*, *LAPACKE_spftri* and *LAPACKE_zpftri*. It also exists with a native Fortran interface as *spftri*.

4.19.1516 LAPACKE_spftri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spftri_work(armpl_int_t matrix_layout, char transr,
                                char uplo, armpl_int_t n, float *a);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpftri_work*, *LAPACKE_dpftri_work*, *LAPACKE_spftri_work* and *LAPACKE_zpftri_work*.

4.19.1517 LAPACKE_spftrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spftrs(armpl_int_t matrix_layout, char transr, char uplo,
                           armpl_int_t n, armpl_int_t nrhs, const float *a,
                           float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpftrs*, *LAPACKE_dpftrs*, *LAPACKE_spftrs* and *LAPACKE_zpftrs*. It also exists with a native Fortran interface as *spftrs*.

4.19.1518 LAPACKE_spftrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spftrs_work(armpl_int_t matrix_layout, char transr,
                                char uplo, armpl_int_t n, armpl_int_t nrhs,
                                const float *a, float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpftrs_work*, *LAPACKE_dpftrs_work*, *LAPACKE_spftrs_work* and *LAPACKE_zpftrs_work*.

4.19.1519 LAPACKE_spocon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spocon(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, const float *a, armpl_int_t lda,
                           float anorm, float *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpocon*, *LAPACKE_dpocon*, *LAPACKE_spocon* and *LAPACKE_zpocon*. It also exists with a native Fortran interface as *spocon*.

4.19.1520 LAPACKE_spocon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spocon_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, const float *a,
                                armpl_int_t lda, float anorm, float *rcond,
                                float *work, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpocon_work*, *LAPACKE_dpocon_work*, *LAPACKE_spocon_work* and *LAPACKE_zpocon_work*.

4.19.1521 LAPACKE_spoequ

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spoequ(armpl_int_t matrix_layout, armpl_int_t n,
                           const float *a, armpl_int_t lda, float *s,
                           float *scond, float *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpoequ*, *LAPACKE_dpoequ*, *LAPACKE_spoequ* and *LAPACKE_zpoequ*. It also exists with a native Fortran interface as *spoequ*.

4.19.1522 LAPACKE_spoequ_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spoequ_work(armpl_int_t matrix_layout, armpl_int_t n,
                                const float *a, armpl_int_t lda, float *s,
                                float *scond, float *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpoequ_work*, *LAPACKE_dpoequ_work*, *LAPACKE_spoequ_work* and *LAPACKE_zpoequ_work*.

4.19.1523 LAPACKE_spoequb

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spoequb(armpl_int_t matrix_layout, armpl_int_t n,
                            const float *a, armpl_int_t lda, float *s,
                            float *scond, float *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpoequb*, *LAPACKE_dpoequb*, *LAPACKE_spoequb* and *LAPACKE_zpoequb*. It also exists with a native Fortran interface as *spoequb*.

4.19.1524 LAPACKE_spoequb_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spoequb_work(armpl_int_t matrix_layout, armpl_int_t n,
                                 const float *a, armpl_int_t lda, float *s,
                                 float *scond, float *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpoequb_work*, *LAPACKE_dpoequb_work*, *LAPACKE_spoequb_work* and *LAPACKE_zpoequb_work*.

4.19.1525 LAPACKE_sporfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sporfs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs, const float *a,
                           armpl_int_t lda, const float *af, armpl_int_t ldaf,
                           const float *b, armpl_int_t ldb, float *x,
                           armpl_int_t ldx, float *ferr, float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cporfs*, *LAPACKE_dporfs*, *LAPACKE_sporfs* and *LAPACKE_zporfs*. It also exists with a native Fortran interface as *sporfs*.

4.19.1526 LAPACKE_sporfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sporfs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const float *a, armpl_int_t lda,
                                const float *af, armpl_int_t ldaf,
                                const float *b, armpl_int_t ldb, float *x,
                                armpl_int_t ldx, float *ferr, float *berr,
                                float *work, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cporfs_work*, *LAPACKE_dporfs_work*, *LAPACKE_sporfs_work* and *LAPACKE_zporfs_work*.

4.19.1527 LAPACKE_sporfsx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sporfsx(armpl_int_t matrix_layout, char uplo, char equed,
                            armpl_int_t n, armpl_int_t nrhs, const float *a,
                            armpl_int_t lda, const float *af,
                            armpl_int_t ldaf, const float *s, const float *b,
                            armpl_int_t ldb, float *x, armpl_int_t ldx,
                            float *rcond, float *berr, armpl_int_t n_err_bnds,
                            float *err_bnds_norm, float *err_bnds_comp,
                            armpl_int_t nparams, float *params);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cporfsx*, *LAPACKE_dporfsx*, *LAPACKE_sporfsx* and *LAPACKE_zporfsx*. It also exists with a native Fortran interface as *sporfsx*.

4.19.1528 LAPACKE_sporfsx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sporfsx_work(armpl_int_t matrix_layout, char uplo,
                                 char equed, armpl_int_t n, armpl_int_t nrhs,
                                 const float *a, armpl_int_t lda,
                                 const float *af, armpl_int_t ldaf,
                                 const float *s, const float *b,
                                 armpl_int_t ldb, float *x, armpl_int_t ldx,
                                 float *rcond, float *berr,
                                 armpl_int_t n_err_bnds, float *err_bnds_norm,
                                 float *err_bnds_comp, armpl_int_t nparams,
                                 float *params, float *work,
                                 armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cporfsx_work*, *LAPACKE_dporfsx_work*, *LAPACKE_sporfsx_work* and *LAPACKE_zporfsx_work*.

4.19.1529 LAPACKE_sposv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sposv(armpl_int_t matrix_layout, char uplo, armpl_int_t n,
                          armpl_int_t nrhs, float *a, armpl_int_t lda,
                          float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cposv*, *LAPACKE_dposv*, *LAPACKE_sposv* and *LAPACKE_zposv*. It also exists with a native Fortran interface as *sposv*.

4.19.1530 LAPACKE_sposv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sposv_work(armpl_int_t matrix_layout, char uplo,
                               armpl_int_t n, armpl_int_t nrhs, float *a,
                               armpl_int_t lda, float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cposv_work*, *LAPACKE_dposv_work*, *LAPACKE_sposv_work* and *LAPACKE_zposv_work*.

4.19.1531 LAPACKE_sposvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sposvx(armpl_int_t matrix_layout, char fact, char uplo,
                           armpl_int_t n, armpl_int_t nrhs, float *a,
                           armpl_int_t lda, float *af, armpl_int_t ldaf,
                           char *equed, float *s, float *b, armpl_int_t ldb,
                           float *x, armpl_int_t ldx, float *rcond,
                           float *ferr, float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cposvx*, *LAPACKE_dposvx*, *LAPACKE_sposvx* and *LAPACKE_zposvx*. It also exists with a native Fortran interface as *sposvx*.

4.19.1532 LAPACKE_sposvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sposvx_work(armpl_int_t matrix_layout, char fact,
                                char uplo, armpl_int_t n, armpl_int_t nrhs,
                                float *a, armpl_int_t lda, float *af,
                                armpl_int_t ldaf, char *equed, float *s,
                                float *b, armpl_int_t ldb, float *x,
                                armpl_int_t ldx, float *rcond, float *ferr,
                                float *berr, float *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cposvx_work*, *LAPACKE_dposvx_work*, *LAPACKE_sposvx_work* and *LAPACKE_zposvx_work*.

4.19.1533 LAPACKE_sposvxx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sposvxx(armpl_int_t matrix_layout, char fact, char uplo,
                            armpl_int_t n, armpl_int_t nrhs, float *a,
                            armpl_int_t lda, float *af, armpl_int_t ldaf,
                            char *equed, float *s, float *b, armpl_int_t ldb,
                            float *x, armpl_int_t ldx, float *rcond,
                            float *rpvgrw, float *berr,
                            armpl_int_t n_err_bnds, float *err_bnds_norm,
                            float *err_bnds_comp, armpl_int_t nparams,
                            float *params);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cposvxx*, *LAPACKE_dposvxx*, *LAPACKE_sposvxx* and *LAPACKE_zposvxx*. It also exists with a native Fortran interface as *sposvxx*.

4.19.1534 LAPACKE_sposvxx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sposvxx_work(armpl_int_t matrix_layout, char fact,
                                 char uplo, armpl_int_t n, armpl_int_t nrhs,
                                 float *a, armpl_int_t lda, float *af,
                                 armpl_int_t ldaf, char *equed, float *s,
                                 float *b, armpl_int_t ldb, float *x,
                                 armpl_int_t ldx, float *rcond, float *rpvgrw,
                                 float *berr, armpl_int_t n_err_bnds,
                                 float *err_bnds_norm, float *err_bnds_comp,
                                 armpl_int_t nparams, float *params,
                                 float *work, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cposvxx_work*, *LAPACKE_dposvxx_work*, *LAPACKE_sposvxx_work* and *LAPACKE_zposvxx_work*.

4.19.1535 LAPACKE_spotrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spotrf(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, float *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpotrf*, *LAPACKE_dpotrf*, *LAPACKE_spotrf* and *LAPACKE_zpotrf*. It also exists with a native Fortran interface as *spotrf*.

4.19.1536 LAPACKE_spotrf2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spotrf2(armpl_int_t matrix_layout, char uplo,
                            armpl_int_t n, float *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpotrf2*, *LAPACKE_dpotrf2*, *LAPACKE_spotrf2* and *LAPACKE_zpotrf2*. It also exists with a native Fortran interface as *spotrf2*.

4.19.1537 LAPACKE_spotrf2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spotrf2_work(armpl_int_t matrix_layout, char uplo,
                                 armpl_int_t n, float *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpotrf2_work*, *LAPACKE_dpotrf2_work*, *LAPACKE_spotrf2_work* and *LAPACKE_zpotrf2_work*.

4.19.1538 LAPACKE_spotrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spotrf_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, float *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpotrf_work*, *LAPACKE_dpotrf_work*, *LAPACKE_spotrf_work* and *LAPACKE_zpotrf_work*.

4.19.1539 LAPACKE_spotri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spotri(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, float *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpotri*, *LAPACKE_dpotri*, *LAPACKE_spotri* and *LAPACKE_zpotri*. It also exists with a native Fortran interface as *spotri*.

4.19.1540 LAPACKE_spotri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spotri_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, float *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpotri_work*, *LAPACKE_dpotri_work*, *LAPACKE_spotri_work* and *LAPACKE_zpotri_work*.

4.19.1541 LAPACKE_spotrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spotrs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs, const float *a,
                           armpl_int_t lda, float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpotrs*, *LAPACKE_dpotrs*, *LAPACKE_spotrs* and *LAPACKE_zpotrs*. It also exists with a native Fortran interface as *spotrs*.

4.19.1542 LAPACKE_spotrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spotrs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const float *a, armpl_int_t lda, float *b,
                                armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpotrs_work*, *LAPACKE_dpotrs_work*, *LAPACKE_spotrs_work* and *LAPACKE_zpotrs_work*.

4.19.1543 LAPACKE_sppcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sppcon(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, const float *ap, float anorm,
                           float *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cppcon*, *LAPACKE_dppcon*, *LAPACKE_sppcon* and *LAPACKE_zppcon*. It also exists with a native Fortran interface as *sppcon*.

4.19.1544 LAPACKE_sppcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sppcon_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, const float *ap, float anorm,
                                float *rcond, float *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cppcon_work*, *LAPACKE_dppcon_work*, *LAPACKE_sppcon_work* and *LAPACKE_zppcon_work*.

4.19.1545 LAPACKE_sppequ

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sppequ(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, const float *ap, float *s,
                           float *scond, float *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cppequ*, *LAPACKE_dppequ*, *LAPACKE_sppequ* and *LAPACKE_zppequ*. It also exists with a native Fortran interface as *sppequ*.

4.19.1546 LAPACKE_sppequ_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sppequ_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, const float *ap, float *s,
                                float *scond, float *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cppequ_work*, *LAPACKE_dppequ_work*, *LAPACKE_sppequ_work* and *LAPACKE_zppequ_work*.

4.19.1547 LAPACKE_spprfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spprfs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs, const float *ap,
                           const float *afp, const float *b, armpl_int_t ldb,
                           float *x, armpl_int_t ldx, float *ferr,
                           float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpprfs*, *LAPACKE_dpprfs*, *LAPACKE_spprfs* and *LAPACKE_zpprfs*. It also exists with a native Fortran interface as *spprfs*.

4.19.1548 LAPACKE_spprfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spprfs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const float *ap, const float *afp,
                                const float *b, armpl_int_t ldb, float *x,
                                armpl_int_t ldx, float *ferr, float *berr,
                                float *work, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpprfs_work*, *LAPACKE_dpprfs_work*, *LAPACKE_spprfs_work* and *LAPACKE_zpprfs_work*.

4.19.1549 LAPACKE_sppsv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sppsv(armpl_int_t matrix_layout, char uplo, armpl_int_t n,
                          armpl_int_t nrhs, float *ap, float *b,
                          armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cppsv*, *LAPACKE_dppsv*, *LAPACKE_sppsv* and *LAPACKE_zppsv*. It also exists with a native Fortran interface as *sppsv*.

4.19.1550 LAPACKE_sppsv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sppsv_work(armpl_int_t matrix_layout, char uplo,
                               armpl_int_t n, armpl_int_t nrhs, float *ap,
                               float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cppsv_work*, *LAPACKE_dppsv_work*, *LAPACKE_sppsv_work* and *LAPACKE_zppsv_work*.

4.19.1551 LAPACKE_sppsvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sppsvx(armpl_int_t matrix_layout, char fact, char uplo,
                           armpl_int_t n, armpl_int_t nrhs, float *ap,
                           float *afp, char *equed, float *s, float *b,
                           armpl_int_t ldb, float *x, armpl_int_t ldx,
                           float *rcond, float *ferr, float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cppsvx*, *LAPACKE_dppsvx*, *LAPACKE_sppsvx* and *LAPACKE_zppsvx*. It also exists with a native Fortran interface as *sppsvx*.

4.19.1552 LAPACKE_sppsvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sppsvx_work(armpl_int_t matrix_layout, char fact,
                                char uplo, armpl_int_t n, armpl_int_t nrhs,
                                float *ap, float *afp, char *equed, float *s,
                                float *b, armpl_int_t ldb, float *x,
                                armpl_int_t ldx, float *rcond, float *ferr,
                                float *berr, float *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cppsvx_work*, *LAPACKE_dppsvx_work*, *LAPACKE_sppsvx_work* and *LAPACKE_zppsvx_work*.

4.19.1553 LAPACKE_spptrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spptrf(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, float *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpptrf*, *LAPACKE_dpptrf*, *LAPACKE_spptrf* and *LAPACKE_zpptrf*. It also exists with a native Fortran interface as *spptrf*.

4.19.1554 LAPACKE_spptrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spptrf_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, float *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpptrf_work*, *LAPACKE_dpptrf_work*, *LAPACKE_spptrf_work* and *LAPACKE_zpptrf_work*.

4.19.1555 LAPACKE_spptri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spptri(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, float *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpptri*, *LAPACKE_dpptri*, *LAPACKE_spptri* and *LAPACKE_zpptri*. It also exists with a native Fortran interface as *spptri*.

4.19.1556 LAPACKE_spptri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spptri_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, float *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpptri_work*, *LAPACKE_dpptri_work*, *LAPACKE_spptri_work* and *LAPACKE_zpptri_work*.

4.19.1557 LAPACKE_spptrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spptrs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs, const float *ap,
                           float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpptrs*, *LAPACKE_dpptrs*, *LAPACKE_spptrs* and *LAPACKE_zpptrs*. It also exists with a native Fortran interface as *spptrs*.

4.19.1558 LAPACKE_spptrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spptrs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const float *ap, float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpptrs_work*, *LAPACKE_dpptrs_work*, *LAPACKE_spptrs_work* and *LAPACKE_zpptrs_work*.

4.19.1559 LAPACKE_spstrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spstrf(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, float *a, armpl_int_t lda,
                           armpl_int_t *piv, armpl_int_t *rank, float tol);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpstrf*, *LAPACKE_dpstrf*, *LAPACKE_spstrf* and *LAPACKE_zpstrf*. It also exists with a native Fortran interface as *spstrf*.

4.19.1560 LAPACKE_spstrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spstrf_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, float *a, armpl_int_t lda,
                                armpl_int_t *piv, armpl_int_t *rank,
                                float tol, float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpstrf_work*, *LAPACKE_dpstrf_work*, *LAPACKE_spstrf_work* and *LAPACKE_zpstrf_work*.

4.19.1561 LAPACKE_sptcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sptcon(armpl_int_t n, const float *d, const float *e,
                           float anorm, float *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cptcon*, *LAPACKE_dptcon*, *LAPACKE_sptcon* and *LAPACKE_zptcon*. It also exists with a native Fortran interface as *sptcon*.

4.19.1562 LAPACKE_sptcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sptcon_work(armpl_int_t n, const float *d, const float *e,
                                float anorm, float *rcond, float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cptcon_work*, *LAPACKE_dptcon_work*, *LAPACKE_sptcon_work* and *LAPACKE_zptcon_work*.

4.19.1563 LAPACKE_spteqr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spteqr(armpl_int_t matrix_layout, char compz,
                           armpl_int_t n, float *d, float *e, float *z,
                           armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpteqr*, *LAPACKE_dpteqr*, *LAPACKE_spteqr* and *LAPACKE_zpteqr*. It also exists with a native Fortran interface as *spteqr*.

4.19.1564 LAPACKE_spteqr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spteqr_work(armpl_int_t matrix_layout, char compz,
                                armpl_int_t n, float *d, float *e, float *z,
                                armpl_int_t ldz, float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpteqr_work*, *LAPACKE_dpteqr_work*, *LAPACKE_spteqr_work* and *LAPACKE_zpteqr_work*.

4.19.1565 LAPACKE_sptrfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sptrfs(armpl_int_t matrix_layout, armpl_int_t n,
                           armpl_int_t nrhs, const float *d, const float *e,
                           const float *df, const float *ef, const float *b,
                           armpl_int_t ldb, float *x, armpl_int_t ldx,
                           float *ferr, float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cptrfs*, *LAPACKE_dptrfs*, *LAPACKE_sptrfs* and *LAPACKE_zptrfs*. It also exists with a native Fortran interface as *sptrfs*.

4.19.1566 LAPACKE_sptrfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sptrfs_work(armpl_int_t matrix_layout, armpl_int_t n,
                                armpl_int_t nrhs, const float *d,
                                const float *e, const float *df,
                                const float *ef, const float *b,
                                armpl_int_t ldb, float *x, armpl_int_t ldx,
                                float *ferr, float *berr, float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cptrfs_work*, *LAPACKE_dptrfs_work*, *LAPACKE_sptrfs_work* and *LAPACKE_zptrfs_work*.

4.19.1567 LAPACKE_sptsv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sptsv(armpl_int_t matrix_layout, armpl_int_t n,
                          armpl_int_t nrhs, float *d, float *e, float *b,
                          armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cptsv*, *LAPACKE_dptsv*, *LAPACKE_sptsv* and *LAPACKE_zptsv*. It also exists with a native Fortran interface as *sptsv*.

4.19.1568 LAPACKE_sptsv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sptsv_work(armpl_int_t matrix_layout, armpl_int_t n,
                               armpl_int_t nrhs, float *d, float *e, float *b,
                               armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cptsv_work*, *LAPACKE_dptsv_work*, *LAPACKE_sptsv_work* and *LAPACKE_zptsv_work*.

4.19.1569 LAPACKE_sptsvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sptsvx(armpl_int_t matrix_layout, char fact,
                           armpl_int_t n, armpl_int_t nrhs, const float *d,
                           const float *e, float *df, float *ef,
                           const float *b, armpl_int_t ldb, float *x,
                           armpl_int_t ldx, float *rcond, float *ferr,
                           float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cptsvx*, *LAPACKE_dptsvx*, *LAPACKE_sptsvx* and *LAPACKE_zptsvx*. It also exists with a native Fortran interface as *sptsvx*.

4.19.1570 LAPACKE_sptsvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sptsvx_work(armpl_int_t matrix_layout, char fact,
                                armpl_int_t n, armpl_int_t nrhs,
                                const float *d, const float *e, float *df,
                                float *ef, const float *b, armpl_int_t ldb,
                                float *x, armpl_int_t ldx, float *rcond,
                                float *ferr, float *berr, float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cptsvx_work*, *LAPACKE_dptsvx_work*, *LAPACKE_sptsvx_work* and *LAPACKE_zptsvx_work*.

4.19.1571 LAPACKE_spttrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spttrf(armpl_int_t n, float *d, float *e);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpttrf*, *LAPACKE_dpttrf*, *LAPACKE_spttrf* and *LAPACKE_zpttrf*. It also exists with a native Fortran interface as *spttrf*.

4.19.1572 LAPACKE_spttrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spttrf_work(armpl_int_t n, float *d, float *e);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpttrf_work*, *LAPACKE_dpttrf_work*, *LAPACKE_spttrf_work* and *LAPACKE_zpttrf_work*.

4.19.1573 LAPACKE_spttrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spttrs(armpl_int_t matrix_layout, armpl_int_t n,
                           armpl_int_t nrhs, const float *d, const float *e,
                           float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpttrs*, *LAPACKE_dpttrs*, *LAPACKE_spttrs* and *LAPACKE_zpttrs*. It also exists with a native Fortran interface as *spttrs*.

4.19.1574 LAPACKE_spttrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_spttrs_work(armpl_int_t matrix_layout, armpl_int_t n,
                                armpl_int_t nrhs, const float *d,
                                const float *e, float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpttrs_work*, *LAPACKE_dpttrs_work*, *LAPACKE_spttrs_work* and *LAPACKE_zpttrs_work*.

4.19.1575 LAPACKE_ssbev

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssbev(armpl_int_t matrix_layout, char jobz, char uplo,
                          armpl_int_t n, armpl_int_t kd, float *ab,
                          armpl_int_t ldab, float *w, float *z,
                          armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbev* and *LAPACKE_ssbev*. It also exists with a native Fortran interface as *ssbev*.

4.19.1576 LAPACKE_ssbev_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssbev_2stage(armpl_int_t matrix_layout, char jobz,
                                 char uplo, armpl_int_t n, armpl_int_t kd,
                                 float *ab, armpl_int_t ldab, float *w,
                                 float *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbev_2stage* and *LAPACKE_ssbev_2stage*. It also exists with a native Fortran interface as *ssbev_2stage*.

4.19.1577 LAPACKE_ssbev_2stage_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssbev_2stage_work(armpl_int_t matrix_layout, char jobz,
                                      char uplo, armpl_int_t n,
                                      armpl_int_t kd, float *ab,
                                      armpl_int_t ldab, float *w, float *z,
                                      armpl_int_t ldz, float *work,
                                      armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbev_2stage_work* and *LAPACKE_ssbev_2stage_work*.

4.19.1578 LAPACKE_ssbev_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssbev_work(armpl_int_t matrix_layout, char jobz,
                               char uplo, armpl_int_t n, armpl_int_t kd,
                               float *ab, armpl_int_t ldab, float *w,
                               float *z, armpl_int_t ldz, float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbev_work* and *LAPACKE_ssbev_work*.

4.19.1579 LAPACKE_ssbevd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssbevd(armpl_int_t matrix_layout, char jobz, char uplo,
                           armpl_int_t n, armpl_int_t kd, float *ab,
                           armpl_int_t ldab, float *w, float *z,
                           armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbevd* and *LAPACKE_ssbevd*. It also exists with a native Fortran interface as *ssbevd*.

4.19.1580 LAPACKE_ssbevd_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssbevd_2stage(armpl_int_t matrix_layout, char jobz,
                                  char uplo, armpl_int_t n, armpl_int_t kd,
                                  float *ab, armpl_int_t ldab, float *w,
                                  float *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbevd_2stage* and *LAPACKE_ssbevd_2stage*. It also exists with a native Fortran interface as *ssbevd_2stage*.

4.19.1581 LAPACKE_ssbevd_2stage_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssbevd_2stage_work(armpl_int_t matrix_layout, char jobz,
                                       char uplo, armpl_int_t n,
                                       armpl_int_t kd, float *ab,
                                       armpl_int_t ldab, float *w, float *z,
                                       armpl_int_t ldz, float *work,
                                       armpl_int_t lwork, armpl_int_t *iwork,
                                       armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbevd_2stage_work* and *LAPACKE_ssbevd_2stage_work*.

4.19.1582 LAPACKE_ssbevd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssbevd_work(armpl_int_t matrix_layout, char jobz,
                                char uplo, armpl_int_t n, armpl_int_t kd,
                                float *ab, armpl_int_t ldab, float *w,
                                float *z, armpl_int_t ldz, float *work,
                                armpl_int_t lwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbevd_work* and *LAPACKE_ssbevd_work*.

4.19.1583 LAPACKE_ssbevx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssbevx(armpl_int_t matrix_layout, char jobz, char range,
                           char uplo, armpl_int_t n, armpl_int_t kd,
                           float *ab, armpl_int_t ldab, float *q,
                           armpl_int_t ldq, float vl, float vu,
                           armpl_int_t il, armpl_int_t iu, float abstol,
                           armpl_int_t *m, float *w, float *z,
                           armpl_int_t ldz, armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbevx* and *LAPACKE_ssbevx*. It also exists with a native Fortran interface as *ssbevx*.

4.19.1584 LAPACKE_ssbevx_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssbevx_2stage(armpl_int_t matrix_layout, char jobz,
                                  char range, char uplo, armpl_int_t n,
                                  armpl_int_t kd, float *ab, armpl_int_t ldab,
                                  float *q, armpl_int_t ldq, float vl,
                                  float vu, armpl_int_t il, armpl_int_t iu,
                                  float abstol, armpl_int_t *m, float *w,
                                  float *z, armpl_int_t ldz,
                                  armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbevx_2stage* and *LAPACKE_ssbevx_2stage*. It also exists with a native Fortran interface as *ssbevx_2stage*.

4.19.1585 LAPACKE_ssbevx_2stage_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssbevx_2stage_work(armpl_int_t matrix_layout, char jobz,
                                       char range, char uplo, armpl_int_t n,
                                       armpl_int_t kd, float *ab,
                                       armpl_int_t ldab, float *q,
                                       armpl_int_t ldq, float vl, float vu,
                                       armpl_int_t il, armpl_int_t iu,
                                       float abstol, armpl_int_t *m, float *w,
                                       float *z, armpl_int_t ldz, float *work,
                                       armpl_int_t lwork, armpl_int_t *iwork,
                                       armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbevx_2stage_work* and *LAPACKE_ssbevx_2stage_work*.

4.19.1586 LAPACKE_ssbevx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssbevx_work(armpl_int_t matrix_layout, char jobz,
                                char range, char uplo, armpl_int_t n,
                                armpl_int_t kd, float *ab, armpl_int_t ldab,
                                float *q, armpl_int_t ldq, float vl, float vu,
                                armpl_int_t il, armpl_int_t iu, float abstol,
                                armpl_int_t *m, float *w, float *z,
                                armpl_int_t ldz, float *work,
                                armpl_int_t *iwork, armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbevx_work* and *LAPACKE_ssbevx_work*.

4.19.1587 LAPACKE_ssbgst

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssbgst(armpl_int_t matrix_layout, char vect, char uplo,
                           armpl_int_t n, armpl_int_t ka, armpl_int_t kb,
                           float *ab, armpl_int_t ldab, const float *bb,
                           armpl_int_t ldbb, float *x, armpl_int_t ldx);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbgst* and *LAPACKE_ssbgst*. It also exists with a native Fortran interface as *ssbgst*.

4.19.1588 LAPACKE_ssbgst_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssbgst_work(armpl_int_t matrix_layout, char vect,
                                char uplo, armpl_int_t n, armpl_int_t ka,
                                armpl_int_t kb, float *ab, armpl_int_t ldab,
                                const float *bb, armpl_int_t ldbb, float *x,
                                armpl_int_t ldx, float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbgst_work* and *LAPACKE_ssbgst_work*.

4.19.1589 LAPACKE_ssbgv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssbgv(armpl_int_t matrix_layout, char jobz, char uplo,
                          armpl_int_t n, armpl_int_t ka, armpl_int_t kb,
                          float *ab, armpl_int_t ldab, float *bb,
                          armpl_int_t ldbb, float *w, float *z,
                          armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbgv* and *LAPACKE_ssbgv*. It also exists with a native Fortran interface as *ssbgv*.

4.19.1590 LAPACKE_ssbgv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssbgv_work(armpl_int_t matrix_layout, char jobz,
                               char uplo, armpl_int_t n, armpl_int_t ka,
                               armpl_int_t kb, float *ab, armpl_int_t ldab,
                               float *bb, armpl_int_t ldbb, float *w,
                               float *z, armpl_int_t ldz, float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbgv_work* and *LAPACKE_ssbgv_work*.

4.19.1591 LAPACKE_ssbgvd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssbgvd(armpl_int_t matrix_layout, char jobz, char uplo,
                           armpl_int_t n, armpl_int_t ka, armpl_int_t kb,
                           float *ab, armpl_int_t ldab, float *bb,
                           armpl_int_t ldbb, float *w, float *z,
                           armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbgvd* and *LAPACKE_ssbgvd*. It also exists with a native Fortran interface as *ssbgvd*.

4.19.1592 LAPACKE_ssbgvd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssbgvd_work(armpl_int_t matrix_layout, char jobz,
                                char uplo, armpl_int_t n, armpl_int_t ka,
                                armpl_int_t kb, float *ab, armpl_int_t ldab,
                                float *bb, armpl_int_t ldbb, float *w,
                                float *z, armpl_int_t ldz, float *work,
                                armpl_int_t lwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbgvd_work* and *LAPACKE_ssbgvd_work*.

4.19.1593 LAPACKE_ssbgvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssbgvx(armpl_int_t matrix_layout, char jobz, char range,
                           char uplo, armpl_int_t n, armpl_int_t ka,
                           armpl_int_t kb, float *ab, armpl_int_t ldab,
                           float *bb, armpl_int_t ldbb, float *q,
                           armpl_int_t ldq, float vl, float vu,
                           armpl_int_t il, armpl_int_t iu, float abstol,
                           armpl_int_t *m, float *w, float *z,
                           armpl_int_t ldz, armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbgvx* and *LAPACKE_ssbgvx*. It also exists with a native Fortran interface as *ssbgvx*.

4.19.1594 LAPACKE_ssbgvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssbgvx_work(armpl_int_t matrix_layout, char jobz,
                                char range, char uplo, armpl_int_t n,
                                armpl_int_t ka, armpl_int_t kb, float *ab,
                                armpl_int_t ldab, float *bb, armpl_int_t ldbb,
                                float *q, armpl_int_t ldq, float vl, float vu,
                                armpl_int_t il, armpl_int_t iu, float abstol,
                                armpl_int_t *m, float *w, float *z,
                                armpl_int_t ldz, float *work,
                                armpl_int_t *iwork, armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbgvx_work* and *LAPACKE_ssbgvx_work*.

4.19.1595 LAPACKE_ssbtrd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssbtrd(armpl_int_t matrix_layout, char vect, char uplo,
                           armpl_int_t n, armpl_int_t kd, float *ab,
                           armpl_int_t ldab, float *d, float *e, float *q,
                           armpl_int_t ldq);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbtrd* and *LAPACKE_ssbtrd*. It also exists with a native Fortran interface as *ssbtrd*.

4.19.1596 LAPACKE_ssbtrd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssbtrd_work(armpl_int_t matrix_layout, char vect,
                                char uplo, armpl_int_t n, armpl_int_t kd,
                                float *ab, armpl_int_t ldab, float *d,
                                float *e, float *q, armpl_int_t ldq,
                                float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsbtrd_work* and *LAPACKE_ssbtrd_work*.

4.19.1597 LAPACKE_ssfrk

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssfrk(armpl_int_t matrix_layout, char transr, char uplo,
                          char trans, armpl_int_t n, armpl_int_t k,
                          float alpha, const float *a, armpl_int_t lda,
                          float beta, float *c);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsfrk* and *LAPACKE_ssfrk*. It also exists with a native Fortran interface as *ssfrk*.

4.19.1598 LAPACKE_ssfrk_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssfrk_work(armpl_int_t matrix_layout, char transr,
                               char uplo, char trans, armpl_int_t n,
                               armpl_int_t k, float alpha, const float *a,
                               armpl_int_t lda, float beta, float *c);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsfrk_work* and *LAPACKE_ssfrk_work*.

4.19.1599 LAPACKE_sspcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sspcon(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, const float *ap,
                           const armpl_int_t *ipiv, float anorm,
                           float *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cspcon*, *LAPACKE_dspcon*, *LAPACKE_sspcon* and *LAPACKE_zspcon*. It also exists with a native Fortran interface as *sspcon*.

4.19.1600 LAPACKE_sspcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sspcon_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, const float *ap,
                                const armpl_int_t *ipiv, float anorm,
                                float *rcond, float *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cspcon_work*, *LAPACKE_dspcon_work*, *LAPACKE_sspcon_work* and *LAPACKE_zspcon_work*.

4.19.1601 LAPACKE_sspev

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sspev(armpl_int_t matrix_layout, char jobz, char uplo,
                          armpl_int_t n, float *ap, float *w, float *z,
                          armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.1602 LAPACKE_sspev_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sspev_work(armpl_int_t matrix_layout, char jobz,
                               char uplo, armpl_int_t n, float *ap, float *w,
                               float *z, armpl_int_t ldz, float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dspev_work* and *LAPACKE_sspev_work*.

4.19.1603 LAPACKE_sspevd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sspevd(armpl_int_t matrix_layout, char jobz, char uplo,
                           armpl_int_t n, float *ap, float *w, float *z,
                           armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dspevd* and *LAPACKE_sspevd*. It also exists with a native Fortran interface as *sspevd*.

4.19.1604 LAPACKE_sspevd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sspevd_work(armpl_int_t matrix_layout, char jobz,
                                char uplo, armpl_int_t n, float *ap, float *w,
                                float *z, armpl_int_t ldz, float *work,
                                armpl_int_t lwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dspevd_work* and *LAPACKE_sspevd_work*.

4.19.1605 LAPACKE_sspevx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sspevx(armpl_int_t matrix_layout, char jobz, char range,
                           char uplo, armpl_int_t n, float *ap, float vl,
                           float vu, armpl_int_t il, armpl_int_t iu,
                           float abstol, armpl_int_t *m, float *w, float *z,
                           armpl_int_t ldz, armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dspevx* and *LAPACKE_sspevx*. It also exists with a native Fortran interface as *sspevx*.

4.19.1606 LAPACKE_sspevx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sspevx_work(armpl_int_t matrix_layout, char jobz,
                                char range, char uplo, armpl_int_t n,
                                float *ap, float vl, float vu, armpl_int_t il,
                                armpl_int_t iu, float abstol, armpl_int_t *m,
                                float *w, float *z, armpl_int_t ldz,
                                float *work, armpl_int_t *iwork,
                                armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dspevx_work* and *LAPACKE_sspevx_work*.

4.19.1607 LAPACKE_sspgst

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sspgst(armpl_int_t matrix_layout, armpl_int_t itype,
                           char uplo, armpl_int_t n, float *ap,
                           const float *bp);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dspgst* and *LAPACKE_sspgst*. It also exists with a native Fortran interface as *sspgst*.

4.19.1608 LAPACKE_sspgst_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sspgst_work(armpl_int_t matrix_layout, armpl_int_t itype,
                                char uplo, armpl_int_t n, float *ap,
                                const float *bp);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dspgst_work* and *LAPACKE_sspgst_work*.

4.19.1609 LAPACKE_sspgv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sspgv(armpl_int_t matrix_layout, armpl_int_t itype,
                          char jobz, char uplo, armpl_int_t n, float *ap,
                          float *bp, float *w, float *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dspgv* and *LAPACKE_sspgv*. It also exists with a native Fortran interface as *sspgv*.

4.19.1610 LAPACKE_sspgv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sspgv_work(armpl_int_t matrix_layout, armpl_int_t itype,
                               char jobz, char uplo, armpl_int_t n, float *ap,
                               float *bp, float *w, float *z, armpl_int_t ldz,
                               float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dspgv_work* and *LAPACKE_sspgv_work*.

4.19.1611 LAPACKE_sspgvd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sspgvd(armpl_int_t matrix_layout, armpl_int_t itype,
                           char jobz, char uplo, armpl_int_t n, float *ap,
                           float *bp, float *w, float *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dspgvd* and *LAPACKE_sspgvd*. It also exists with a native Fortran interface as *sspgvd*.

4.19.1612 LAPACKE_sspgvd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sspgvd_work(armpl_int_t matrix_layout, armpl_int_t itype,
                                char jobz, char uplo, armpl_int_t n,
                                float *ap, float *bp, float *w, float *z,
                                armpl_int_t ldz, float *work,
                                armpl_int_t lwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dspgvd_work* and *LAPACKE_sspgvd_work*.

4.19.1613 LAPACKE_sspgvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sspgvx(armpl_int_t matrix_layout, armpl_int_t itype,
                           char jobz, char range, char uplo, armpl_int_t n,
                           float *ap, float *bp, float vl, float vu,
                           armpl_int_t il, armpl_int_t iu, float abstol,
                           armpl_int_t *m, float *w, float *z,
                           armpl_int_t ldz, armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dspgvx* and *LAPACKE_sspgvx*. It also exists with a native Fortran interface as *sspgvx*.

4.19.1614 LAPACKE_sspgvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sspgvx_work(armpl_int_t matrix_layout, armpl_int_t itype,
                                char jobz, char range, char uplo,
                                armpl_int_t n, float *ap, float *bp, float vl,
                                float vu, armpl_int_t il, armpl_int_t iu,
                                float abstol, armpl_int_t *m, float *w,
                                float *z, armpl_int_t ldz, float *work,
                                armpl_int_t *iwork, armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dspgvx_work* and *LAPACKE_sspgvx_work*.

4.19.1615 LAPACKE_ssprfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssprfs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs, const float *ap,
                           const float *afp, const armpl_int_t *ipiv,
                           const float *b, armpl_int_t ldb, float *x,
                           armpl_int_t ldx, float *ferr, float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csprfs*, *LAPACKE_dsprfs*, *LAPACKE_ssprfs* and *LAPACKE_zsprfs*. It also exists with a native Fortran interface as *ssprfs*.

4.19.1616 LAPACKE_ssprfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssprfs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const float *ap, const float *afp,
                                const armpl_int_t *ipiv, const float *b,
                                armpl_int_t ldb, float *x, armpl_int_t ldx,
                                float *ferr, float *berr, float *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csprfs_work*, *LAPACKE_dsprfs_work*, *LAPACKE_ssprfs_work* and *LAPACKE_zsprfs_work*.

4.19.1617 LAPACKE_sspsv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sspsv(armpl_int_t matrix_layout, char uplo, armpl_int_t n,
                          armpl_int_t nrhs, float *ap, armpl_int_t *ipiv,
                          float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cspsv*, *LAPACKE_dspsv*, *LAPACKE_sspsv* and *LAPACKE_zspsv*. It also exists with a native Fortran interface as *sspsv*.

4.19.1618 LAPACKE_sspsv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sspsv_work(armpl_int_t matrix_layout, char uplo,
                               armpl_int_t n, armpl_int_t nrhs, float *ap,
                               armpl_int_t *ipiv, float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cspsv_work*, *LAPACKE_dspsv_work*, *LAPACKE_sspsv_work* and *LAPACKE_zspsv_work*.

4.19.1619 LAPACKE_sspsvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sspsvx(armpl_int_t matrix_layout, char fact, char uplo,
                           armpl_int_t n, armpl_int_t nrhs, const float *ap,
                           float *afp, armpl_int_t *ipiv, const float *b,
                           armpl_int_t ldb, float *x, armpl_int_t ldx,
                           float *rcond, float *ferr, float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cspsvx*, *LAPACKE_dspsvx*, *LAPACKE_sspsvx* and *LAPACKE_zspsvx*. It also exists with a native Fortran interface as *sspsvx*.

4.19.1620 LAPACKE_sspsvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sspsvx_work(armpl_int_t matrix_layout, char fact,
                                char uplo, armpl_int_t n, armpl_int_t nrhs,
                                const float *ap, float *afp,
                                armpl_int_t *ipiv, const float *b,
                                armpl_int_t ldb, float *x, armpl_int_t ldx,
                                float *rcond, float *ferr, float *berr,
                                float *work, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cspsvx_work*, *LAPACKE_dspsvx_work*, *LAPACKE_sspsvx_work* and *LAPACKE_zspsvx_work*.

4.19.1621 LAPACKE_ssptrd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssptrd(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, float *ap, float *d, float *e,
                           float *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsptrd* and *LAPACKE_ssptrd*. It also exists with a native Fortran interface as *ssptrd*.

4.19.1622 LAPACKE_ssptrd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssptrd_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, float *ap, float *d, float *e,
                                float *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsptrd_work* and *LAPACKE_ssptrd_work*.

4.19.1623 LAPACKE_ssptrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssptrf(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, float *ap, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csptrf*, *LAPACKE_dsptrf*, *LAPACKE_ssptrf* and *LAPACKE_zsptrf*. It also exists with a native Fortran interface as *ssptrf*.

4.19.1624 LAPACKE_ssptrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssptrf_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, float *ap, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csptrf_work*, *LAPACKE_dsptrf_work*, *LAPACKE_ssptrf_work* and *LAPACKE_zsptrf_work*.

4.19.1625 LAPACKE_ssptri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssptri(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, float *ap,
                           const armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csptri*, *LAPACKE_dsptri*, *LAPACKE_ssptri* and *LAPACKE_zsptri*. It also exists with a native Fortran interface as *ssptri*.

4.19.1626 LAPACKE_ssptri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssptri_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, float *ap,
                                const armpl_int_t *ipiv, float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csptri_work*, *LAPACKE_dsptri_work*, *LAPACKE_ssptri_work* and *LAPACKE_zsptri_work*.

4.19.1627 LAPACKE_ssptrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssptrs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs, const float *ap,
                           const armpl_int_t *ipiv, float *b,
                           armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csptrs*, *LAPACKE_dsptrs*, *LAPACKE_ssptrs* and *LAPACKE_zsptrs*. It also exists with a native Fortran interface as *ssptrs*.

4.19.1628 LAPACKE_ssptrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssptrs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const float *ap, const armpl_int_t *ipiv,
                                float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csptrs_work*, *LAPACKE_dsptrs_work*, *LAPACKE_ssptrs_work* and *LAPACKE_zsptrs_work*.

4.19.1629 LAPACKE_sstebz

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sstebz(char range, char order, armpl_int_t n, float vl,
                           float vu, armpl_int_t il, armpl_int_t iu,
                           float abstol, const float *d, const float *e,
                           armpl_int_t *m, armpl_int_t *nsplit, float *w,
                           armpl_int_t *iblock, armpl_int_t *isplit);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.1630 LAPACKE_sstebz_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sstebz_work(char range, char order, armpl_int_t n,
                                float vl, float vu, armpl_int_t il,
                                armpl_int_t iu, float abstol, const float *d,
                                const float *e, armpl_int_t *m,
                                armpl_int_t *nsplit, float *w,
                                armpl_int_t *iblock, armpl_int_t *isplit,
                                float *work, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dstebz_work* and *LAPACKE_sstebz_work*.

4.19.1631 LAPACKE_sstedc

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sstedc(armpl_int_t matrix_layout, char compz,
                           armpl_int_t n, float *d, float *e, float *z,
                           armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cstedc*, *LAPACKE_dstedc*, *LAPACKE_sstedc* and *LAPACKE_zstedc*. It also exists with a native Fortran interface as *sstedc*.

4.19.1632 LAPACKE_sstedc_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sstedc_work(armpl_int_t matrix_layout, char compz,
                                armpl_int_t n, float *d, float *e, float *z,
                                armpl_int_t ldz, float *work,
                                armpl_int_t lwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cstedc_work*, *LAPACKE_dstedc_work*, *LAPACKE_sstedc_work* and *LAPACKE_zstedc_work*.

4.19.1633 LAPACKE_sstegr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sstegr(armpl_int_t matrix_layout, char jobz, char range,
                           armpl_int_t n, float *d, float *e, float vl,
                           float vu, armpl_int_t il, armpl_int_t iu,
                           float abstol, armpl_int_t *m, float *w, float *z,
                           armpl_int_t ldz, armpl_int_t *isuppz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cstegr*, *LAPACKE_dstegr*, *LAPACKE_sstegr* and *LAPACKE_zstegr*. It also exists with a native Fortran interface as *sstegr*.

4.19.1634 LAPACKE_sstegr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sstegr_work(armpl_int_t matrix_layout, char jobz,
                                char range, armpl_int_t n, float *d, float *e,
                                float vl, float vu, armpl_int_t il,
                                armpl_int_t iu, float abstol, armpl_int_t *m,
                                float *w, float *z, armpl_int_t ldz,
                                armpl_int_t *isuppz, float *work,
                                armpl_int_t lwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cstegr_work*, *LAPACKE_dstegr_work*, *LAPACKE_sstegr_work* and *LAPACKE_zstegr_work*.

4.19.1635 LAPACKE_sstein

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sstein(armpl_int_t matrix_layout, armpl_int_t n,
                           const float *d, const float *e, armpl_int_t m,
                           const float *w, const armpl_int_t *iblock,
                           const armpl_int_t *isplit, float *z,
                           armpl_int_t ldz, armpl_int_t *ifailv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cstein*, *LAPACKE_dstein*, *LAPACKE_sstein* and *LAPACKE_zstein*. It also exists with a native Fortran interface as *sstein*.

4.19.1636 LAPACKE_sstein_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sstein_work(armpl_int_t matrix_layout, armpl_int_t n,
                                const float *d, const float *e, armpl_int_t m,
                                const float *w, const armpl_int_t *iblock,
                                const armpl_int_t *isplit, float *z,
                                armpl_int_t ldz, float *work,
                                armpl_int_t *iwork, armpl_int_t *ifailv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cstein_work*, *LAPACKE_dstein_work*, *LAPACKE_sstein_work* and *LAPACKE_zstein_work*.

4.19.1637 LAPACKE_sstemr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sstemr(armpl_int_t matrix_layout, char jobz, char range,
                           armpl_int_t n, float *d, float *e, float vl,
                           float vu, armpl_int_t il, armpl_int_t iu,
                           armpl_int_t *m, float *w, float *z,
                           armpl_int_t ldz, armpl_int_t nzc,
                           armpl_int_t *isuppz, armpl_int_t *tryrac);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cstemr*, *LAPACKE_dstemr*, *LAPACKE_sstemr* and *LAPACKE_zstemr*. It also exists with a native Fortran interface as *sstemr*.

4.19.1638 LAPACKE_sstemr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sstemr_work(armpl_int_t matrix_layout, char jobz,
                                char range, armpl_int_t n, float *d, float *e,
                                float vl, float vu, armpl_int_t il,
                                armpl_int_t iu, armpl_int_t *m, float *w,
                                float *z, armpl_int_t ldz, armpl_int_t nzc,
                                armpl_int_t *isuppz, armpl_int_t *tryrac,
                                float *work, armpl_int_t lwork,
                                armpl_int_t *iwork, armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cstemr_work*, *LAPACKE_dstemr_work*, *LAPACKE_sstemr_work* and *LAPACKE_zstemr_work*.

4.19.1639 LAPACKE_ssteqr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssteqr(armpl_int_t matrix_layout, char compz,
                           armpl_int_t n, float *d, float *e, float *z,
                           armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csteqr*, *LAPACKE_dsteqr*, *LAPACKE_ssteqr* and *LAPACKE_zsteqr*. It also exists with a native Fortran interface as *ssteqr*.

4.19.1640 LAPACKE_ssteqr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssteqr_work(armpl_int_t matrix_layout, char compz,
                                armpl_int_t n, float *d, float *e, float *z,
                                armpl_int_t ldz, float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csteqr_work*, *LAPACKE_dsteqr_work*, *LAPACKE_ssteqr_work* and *LAPACKE_zsteqr_work*.

4.19.1641 LAPACKE_ssterf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssterf(armpl_int_t n, float *d, float *e);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsterf* and *LAPACKE_ssterf*. It also exists with a native Fortran interface as *ssterf*.

4.19.1642 LAPACKE_ssterf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssterf_work(armpl_int_t n, float *d, float *e);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsterf_work* and *LAPACKE_ssterf_work*.

4.19.1643 LAPACKE_sstev

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sstev(armpl_int_t matrix_layout, char jobz, armpl_int_t n,
                          float *d, float *e, float *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dstev* and *LAPACKE_sstev*. It also exists with a native Fortran interface as *sstev*.

4.19.1644 LAPACKE_sstev_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sstev_work(armpl_int_t matrix_layout, char jobz,
                               armpl_int_t n, float *d, float *e, float *z,
                               armpl_int_t ldz, float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dstev_work* and *LAPACKE_sstev_work*.

4.19.1645 LAPACKE_sstevd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sstevd(armpl_int_t matrix_layout, char jobz,
                           armpl_int_t n, float *d, float *e, float *z,
                           armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dstevd* and *LAPACKE_sstevd*. It also exists with a native Fortran interface as *sstevd*.

4.19.1646 LAPACKE_sstevd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sstevd_work(armpl_int_t matrix_layout, char jobz,
                                armpl_int_t n, float *d, float *e, float *z,
                                armpl_int_t ldz, float *work,
                                armpl_int_t lwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dstevd_work* and *LAPACKE_sstevd_work*.

4.19.1647 LAPACKE_sstevr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sstevr(armpl_int_t matrix_layout, char jobz, char range,
                           armpl_int_t n, float *d, float *e, float vl,
                           float vu, armpl_int_t il, armpl_int_t iu,
                           float abstol, armpl_int_t *m, float *w, float *z,
                           armpl_int_t ldz, armpl_int_t *isuppz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dstevr* and *LAPACKE_sstevr*. It also exists with a native Fortran interface as *sstevr*.

4.19.1648 LAPACKE_sstevr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sstevr_work(armpl_int_t matrix_layout, char jobz,
                                char range, armpl_int_t n, float *d, float *e,
                                float vl, float vu, armpl_int_t il,
                                armpl_int_t iu, float abstol, armpl_int_t *m,
                                float *w, float *z, armpl_int_t ldz,
                                armpl_int_t *isuppz, float *work,
                                armpl_int_t lwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dstevr_work* and *LAPACKE_sstevr_work*.

4.19.1649 LAPACKE_sstevx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sstevx(armpl_int_t matrix_layout, char jobz, char range,
                           armpl_int_t n, float *d, float *e, float vl,
                           float vu, armpl_int_t il, armpl_int_t iu,
                           float abstol, armpl_int_t *m, float *w, float *z,
                           armpl_int_t ldz, armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dstevx* and *LAPACKE_sstevx*. It also exists with a native Fortran interface as *sstevx*.

4.19.1650 LAPACKE_sstevx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_sstevx_work(armpl_int_t matrix_layout, char jobz,
                                char range, armpl_int_t n, float *d, float *e,
                                float vl, float vu, armpl_int_t il,
                                armpl_int_t iu, float abstol, armpl_int_t *m,
                                float *w, float *z, armpl_int_t ldz,
                                float *work, armpl_int_t *iwork,
                                armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dstevx_work* and *LAPACKE_sstevx_work*.

4.19.1651 LAPACKE_ssycon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssycon(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, const float *a, armpl_int_t lda,
                           const armpl_int_t *ipiv, float anorm,
                           float *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csycon*, *LAPACKE_dsycon*, *LAPACKE_ssycon* and *LAPACKE_zsycon*. It also exists with a native Fortran interface as *ssycon*.

4.19.1652 LAPACKE_ssycon_3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssycon_3(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, const float *a, armpl_int_t lda,
                             const float *e, const armpl_int_t *ipiv,
                             float anorm, float *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csycon_3*, *LAPACKE_dsycon_3*, *LAPACKE_ssycon_3* and *LAPACKE_zsycon_3*. It also exists with a native Fortran interface as *ssycon_3*.

4.19.1653 LAPACKE_ssycon_3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssycon_3_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, const float *a,
                                  armpl_int_t lda, const float *e,
                                  const armpl_int_t *ipiv, float anorm,
                                  float *rcond, float *work,
                                  armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csycon_3_work*, *LAPACKE_dsycon_3_work*, *LAPACKE_ssycon_3_work* and *LAPACKE_zsycon_3_work*.

4.19.1654 LAPACKE_ssycon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssycon_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, const float *a,
                                armpl_int_t lda, const armpl_int_t *ipiv,
                                float anorm, float *rcond, float *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csycon_work*, *LAPACKE_dsycon_work*, *LAPACKE_ssycon_work* and *LAPACKE_zsycon_work*.

4.19.1655 LAPACKE_ssyconv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssyconv(armpl_int_t matrix_layout, char uplo, char way,
                            armpl_int_t n, float *a, armpl_int_t lda,
                            const armpl_int_t *ipiv, float *e);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyconv*, *LAPACKE_dsyconv*, *LAPACKE_ssyconv* and *LAPACKE_zsyconv*. It also exists with a native Fortran interface as *ssyconv*.

4.19.1656 LAPACKE_ssyconv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssyconv_work(armpl_int_t matrix_layout, char uplo,
                                 char way, armpl_int_t n, float *a,
                                 armpl_int_t lda, const armpl_int_t *ipiv,
                                 float *e);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyconv_work*, *LAPACKE_dsyconv_work*, *LAPACKE_ssyconv_work* and *LAPACKE_zsyconv_work*.

4.19.1657 LAPACKE_ssyequb

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssyequb(armpl_int_t matrix_layout, char uplo,
                            armpl_int_t n, const float *a, armpl_int_t lda,
                            float *s, float *scond, float *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyequb*, *LAPACKE_dsyequb*, *LAPACKE_ssyequb* and *LAPACKE_zsyequb*. It also exists with a native Fortran interface as *ssyequb*.

4.19.1658 LAPACKE_ssyequb_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssyequb_work(armpl_int_t matrix_layout, char uplo,
                                 armpl_int_t n, const float *a,
                                 armpl_int_t lda, float *s, float *scond,
                                 float *amax, float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyequb_work*, *LAPACKE_dsyequb_work*, *LAPACKE_ssyequb_work* and *LAPACKE_zsyequb_work*.

4.19.1659 LAPACKE_ssyev

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssyev(armpl_int_t matrix_layout, char jobz, char uplo,
                          armpl_int_t n, float *a, armpl_int_t lda,
                          float *w);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsyev* and *LAPACKE_ssyev*. It also exists with a native Fortran interface as *ssyev*.

4.19.1660 LAPACKE_ssyev_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssyev_2stage(armpl_int_t matrix_layout, char jobz,
                                 char uplo, armpl_int_t n, float *a,
                                 armpl_int_t lda, float *w);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsyev_2stage* and *LAPACKE_ssyev_2stage*. It also exists with a native Fortran interface as *ssyev_2stage*.

4.19.1661 LAPACKE_ssyev_2stage_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssyev_2stage_work(armpl_int_t matrix_layout, char jobz,
                                      char uplo, armpl_int_t n, float *a,
                                      armpl_int_t lda, float *w, float *work,
                                      armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsyev_2stage_work* and *LAPACKE_ssyev_2stage_work*.

4.19.1662 LAPACKE_ssyev_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssyev_work(armpl_int_t matrix_layout, char jobz,
                               char uplo, armpl_int_t n, float *a,
                               armpl_int_t lda, float *w, float *work,
                               armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsyev_work* and *LAPACKE_ssyev_work*.

4.19.1663 LAPACKE_ssyevd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssyevd(armpl_int_t matrix_layout, char jobz, char uplo,
                           armpl_int_t n, float *a, armpl_int_t lda,
                           float *w);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsyevd* and *LAPACKE_ssyevd*. It also exists with a native Fortran interface as *ssyevd*.

4.19.1664 LAPACKE_ssyevd_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssyevd_2stage(armpl_int_t matrix_layout, char jobz,
                                  char uplo, armpl_int_t n, float *a,
                                  armpl_int_t lda, float *w);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsyevd_2stage* and *LAPACKE_ssyevd_2stage*. It also exists with a native Fortran interface as *ssyevd_2stage*.

4.19.1665 LAPACKE_ssyevd_2stage_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssyevd_2stage_work(armpl_int_t matrix_layout, char jobz,
                                       char uplo, armpl_int_t n, float *a,
                                       armpl_int_t lda, float *w, float *work,
                                       armpl_int_t lwork, armpl_int_t *iwork,
                                       armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsyevd_2stage_work* and *LAPACKE_ssyevd_2stage_work*.

4.19.1666 LAPACKE_ssyevd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssyevd_work(armpl_int_t matrix_layout, char jobz,
                                char uplo, armpl_int_t n, float *a,
                                armpl_int_t lda, float *w, float *work,
                                armpl_int_t lwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsyevd_work* and *LAPACKE_ssyevd_work*.

4.19.1667 LAPACKE_ssyevr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssyevr(armpl_int_t matrix_layout, char jobz, char range,
                           char uplo, armpl_int_t n, float *a,
                           armpl_int_t lda, float vl, float vu,
                           armpl_int_t il, armpl_int_t iu, float abstol,
                           armpl_int_t *m, float *w, float *z,
                           armpl_int_t ldz, armpl_int_t *isuppz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsyevr* and *LAPACKE_ssyevr*. It also exists with a native Fortran interface as *ssyevr*.

4.19.1668 LAPACKE_ssyevr_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssyevr_2stage(armpl_int_t matrix_layout, char jobz,
                                  char range, char uplo, armpl_int_t n,
                                  float *a, armpl_int_t lda, float vl,
                                  float vu, armpl_int_t il, armpl_int_t iu,
```

(continues on next page)

(continued from previous page)

```
                                    float abstol, armpl_int_t *m, float *w,
                                    float *z, armpl_int_t ldz,
                                    armpl_int_t *isuppz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsyevr_2stage* and *LAPACKE_ssyevr_2stage*. It also exists with a native Fortran interface as *ssyevr_2stage*.

4.19.1669 LAPACKE_ssyevr_2stage_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssyevr_2stage_work(armpl_int_t matrix_layout, char jobz,
                                       char range, char uplo, armpl_int_t n,
                                       float *a, armpl_int_t lda, float vl,
                                       float vu, armpl_int_t il,
                                       armpl_int_t iu, float abstol,
                                       armpl_int_t *m, float *w, float *z,
                                       armpl_int_t ldz, armpl_int_t *isuppz,
                                       float *work, armpl_int_t lwork,
                                       armpl_int_t *iwork,
                                       armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsyevr_2stage_work* and *LAPACKE_ssyevr_2stage_work*.

4.19.1670 LAPACKE_ssyevr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssyevr_work(armpl_int_t matrix_layout, char jobz,
                                char range, char uplo, armpl_int_t n,
                                float *a, armpl_int_t lda, float vl, float vu,
                                armpl_int_t il, armpl_int_t iu, float abstol,
                                armpl_int_t *m, float *w, float *z,
                                armpl_int_t ldz, armpl_int_t *isuppz,
                                float *work, armpl_int_t lwork,
                                armpl_int_t *iwork, armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsyevr_work* and *LAPACKE_ssyevr_work*.

4.19.1671 LAPACKE_ssyevx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssyevx(armpl_int_t matrix_layout, char jobz, char range,
                           char uplo, armpl_int_t n, float *a,
                           armpl_int_t lda, float vl, float vu,
                           armpl_int_t il, armpl_int_t iu, float abstol,
                           armpl_int_t *m, float *w, float *z,
                           armpl_int_t ldz, armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsyevx* and *LAPACKE_ssyevx*. It also exists with a native Fortran interface as *ssyevx*.

4.19.1672 LAPACKE_ssyevx_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssyevx_2stage(armpl_int_t matrix_layout, char jobz,
                                  char range, char uplo, armpl_int_t n,
                                  float *a, armpl_int_t lda, float vl,
                                  float vu, armpl_int_t il, armpl_int_t iu,
                                  float abstol, armpl_int_t *m, float *w,
                                  float *z, armpl_int_t ldz,
                                  armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsyevx_2stage* and *LAPACKE_ssyevx_2stage*. It also exists with a native Fortran interface as *ssyevx_2stage*.

4.19.1673 LAPACKE_ssyevx_2stage_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssyevx_2stage_work(armpl_int_t matrix_layout, char jobz,
                                       char range, char uplo, armpl_int_t n,
                                       float *a, armpl_int_t lda, float vl,
                                       float vu, armpl_int_t il,
                                       armpl_int_t iu, float abstol,
                                       armpl_int_t *m, float *w, float *z,
                                       armpl_int_t ldz, float *work,
                                       armpl_int_t lwork, armpl_int_t *iwork,
                                       armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.1674 LAPACKE_ssyevx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssyevx_work(armpl_int_t matrix_layout, char jobz,
                                char range, char uplo, armpl_int_t n,
                                float *a, armpl_int_t lda, float vl, float vu,
                                armpl_int_t il, armpl_int_t iu, float abstol,
                                armpl_int_t *m, float *w, float *z,
                                armpl_int_t ldz, float *work,
                                armpl_int_t lwork, armpl_int_t *iwork,
                                armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsyevx_work* and *LAPACKE_ssyevx_work*.

4.19.1675 LAPACKE_ssygst

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssygst(armpl_int_t matrix_layout, armpl_int_t itype,
                           char uplo, armpl_int_t n, float *a,
                           armpl_int_t lda, const float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsygst* and *LAPACKE_ssygst*. It also exists with a native Fortran interface as *ssygst*.

4.19.1676 LAPACKE_ssygst_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssygst_work(armpl_int_t matrix_layout, armpl_int_t itype,
                                char uplo, armpl_int_t n, float *a,
                                armpl_int_t lda, const float *b,
                                armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsygst_work* and *LAPACKE_ssygst_work*.

4.19.1677 LAPACKE_ssygv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssygv(armpl_int_t matrix_layout, armpl_int_t itype,
                          char jobz, char uplo, armpl_int_t n, float *a,
                          armpl_int_t lda, float *b, armpl_int_t ldb,
                          float *w);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsygv* and *LAPACKE_ssygv*. It also exists with a native Fortran interface as *ssygv*.

4.19.1678 LAPACKE_ssygv_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssygv_2stage(armpl_int_t matrix_layout, armpl_int_t itype,
                                 char jobz, char uplo, armpl_int_t n,
                                 float *a, armpl_int_t lda, float *b,
                                 armpl_int_t ldb, float *w);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsygv_2stage* and *LAPACKE_ssygv_2stage*. It also exists with a native Fortran interface as *ssygv_2stage*.

4.19.1679 LAPACKE_ssygv_2stage_work

4.19.1680 LAPACKE_ssygv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssygv_work(armpl_int_t matrix_layout, armpl_int_t itype,
                               char jobz, char uplo, armpl_int_t n, float *a,
```

(continues on next page)

(continued from previous page)

```
                                armpl_int_t lda, float *b, armpl_int_t ldb,
                                float *w, float *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsygv_work* and *LAPACKE_ssygv_work*.

4.19.1681 LAPACKE_ssygvd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssygvd(armpl_int_t matrix_layout, armpl_int_t itype,
                           char jobz, char uplo, armpl_int_t n, float *a,
                           armpl_int_t lda, float *b, armpl_int_t ldb,
                           float *w);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsygvd* and *LAPACKE_ssygvd*. It also exists with a native Fortran interface as *ssygvd*.

4.19.1682 LAPACKE_ssygvd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssygvd_work(armpl_int_t matrix_layout, armpl_int_t itype,
                                char jobz, char uplo, armpl_int_t n, float *a,
                                armpl_int_t lda, float *b, armpl_int_t ldb,
                                float *w, float *work, armpl_int_t lwork,
                                armpl_int_t *iwork, armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsygvd_work* and *LAPACKE_ssygvd_work*.

4.19.1683 LAPACKE_ssygvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssygvx(armpl_int_t matrix_layout, armpl_int_t itype,
                           char jobz, char range, char uplo, armpl_int_t n,
                           float *a, armpl_int_t lda, float *b,
                           armpl_int_t ldb, float vl, float vu,
                           armpl_int_t il, armpl_int_t iu, float abstol,
                           armpl_int_t *m, float *w, float *z,
                           armpl_int_t ldz, armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsygvx* and *LAPACKE_ssygvx*. It also exists with a native Fortran interface as *ssygvx*.

4.19.1684 LAPACKE_ssygvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssygvx_work(armpl_int_t matrix_layout, armpl_int_t itype,
                                char jobz, char range, char uplo,
                                armpl_int_t n, float *a, armpl_int_t lda,
                                float *b, armpl_int_t ldb, float vl, float vu,
                                armpl_int_t il, armpl_int_t iu, float abstol,
                                armpl_int_t *m, float *w, float *z,
                                armpl_int_t ldz, float *work,
                                armpl_int_t lwork, armpl_int_t *iwork,
                                armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsygvx_work* and *LAPACKE_ssygvx_work*.

4.19.1685 LAPACKE_ssyrfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssyrfs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs, const float *a,
                           armpl_int_t lda, const float *af, armpl_int_t ldaf,
                           const armpl_int_t *ipiv, const float *b,
                           armpl_int_t ldb, float *x, armpl_int_t ldx,
                           float *ferr, float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyrfs*, *LAPACKE_dsyrfs*, *LAPACKE_ssyrfs* and *LAPACKE_zsyrfs*. It also exists with a native Fortran interface as *ssyrfs*.

4.19.1686 LAPACKE_ssyrfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssyrfs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const float *a, armpl_int_t lda,
                                const float *af, armpl_int_t ldaf,
                                const armpl_int_t *ipiv, const float *b,
                                armpl_int_t ldb, float *x, armpl_int_t ldx,
                                float *ferr, float *berr, float *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyrfs_work*, *LAPACKE_dsyrfs_work*, *LAPACKE_ssyrfs_work* and *LAPACKE_zsyrfs_work*.

4.19.1687 LAPACKE_ssyrfsx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssyrfsx(armpl_int_t matrix_layout, char uplo, char equed,
                            armpl_int_t n, armpl_int_t nrhs, const float *a,
                            armpl_int_t lda, const float *af,
                            armpl_int_t ldaf, const armpl_int_t *ipiv,
                            const float *s, const float *b, armpl_int_t ldb,
                            float *x, armpl_int_t ldx, float *rcond,
                            float *berr, armpl_int_t n_err_bnds,
                            float *err_bnds_norm, float *err_bnds_comp,
                            armpl_int_t nparams, float *params);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyrfsx*, *LAPACKE_dsyrfsx*, *LAPACKE_ssyrfsx* and *LAPACKE_zsyrfsx*. It also exists with a native Fortran interface as *ssyrfsx*.

4.19.1688 LAPACKE_ssyrfsx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssyrfsx_work(armpl_int_t matrix_layout, char uplo,
                                 char equed, armpl_int_t n, armpl_int_t nrhs,
                                 const float *a, armpl_int_t lda,
                                 const float *af, armpl_int_t ldaf,
                                 const armpl_int_t *ipiv, const float *s,
                                 const float *b, armpl_int_t ldb, float *x,
                                 armpl_int_t ldx, float *rcond, float *berr,
                                 armpl_int_t n_err_bnds, float *err_bnds_norm,
                                 float *err_bnds_comp, armpl_int_t nparams,
                                 float *params, float *work,
                                 armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyrfsx_work*, *LAPACKE_dsyrfsx_work*, *LAPACKE_ssyrfsx_work* and *LAPACKE_zsyrfsx_work*.

4.19.1689 LAPACKE_ssysv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssysv(armpl_int_t matrix_layout, char uplo, armpl_int_t n,
                          armpl_int_t nrhs, float *a, armpl_int_t lda,
                          armpl_int_t *ipiv, float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysv*, *LAPACKE_dsysv*, *LAPACKE_ssysv* and *LAPACKE_zsysv*. It also exists with a native Fortran interface as *ssysv*.

4.19.1690 LAPACKE_ssysv_aa

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssysv_aa(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, armpl_int_t nrhs, float *a,
                             armpl_int_t lda, armpl_int_t *ipiv, float *b,
                             armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysv_aa*, *LAPACKE_dsysv_aa*, *LAPACKE_ssysv_aa* and *LAPACKE_zsysv_aa*. It also exists with a native Fortran interface as *ssysv_aa*.

4.19.1691 LAPACKE_ssysv_aa_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssysv_aa_2stage(armpl_int_t matrix_layout, char uplo,
                                    armpl_int_t n, armpl_int_t nrhs, float *a,
                                    armpl_int_t lda, float *tb,
                                    armpl_int_t ltb, armpl_int_t *ipiv,
                                    armpl_int_t *ipiv2, float *b,
                                    armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysv_aa_2stage*, *LAPACKE_dsysv_aa_2stage*, *LAPACKE_ssysv_aa_2stage* and *LAPACKE_zsysv_aa_2stage*. It also exists with a native Fortran interface as *ssysv_aa_2stage*.

4.19.1692 LAPACKE_ssysv_aa_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssysv_aa_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, armpl_int_t nrhs, float *a,
                                  armpl_int_t lda, armpl_int_t *ipiv,
                                  float *b, armpl_int_t ldb, float *work,
                                  armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysv_aa_work*, *LAPACKE_dsysv_aa_work*, *LAPACKE_ssysv_aa_work* and *LAPACKE_zsysv_aa_work*.

4.19.1693 LAPACKE_ssysv_rk

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssysv_rk(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, armpl_int_t nrhs, float *a,
                             armpl_int_t lda, float *e, armpl_int_t *ipiv,
                             float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysv_rk*, *LAPACKE_dsysv_rk*, *LAPACKE_ssysv_rk* and *LAPACKE_zsysv_rk*. It also exists with a native Fortran interface as *ssysv_rk*.

4.19.1694 LAPACKE_ssysv_rk_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssysv_rk_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, armpl_int_t nrhs, float *a,
                                  armpl_int_t lda, float *e,
                                  armpl_int_t *ipiv, float *b,
                                  armpl_int_t ldb, float *work,
                                  armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysv_rk_work*, *LAPACKE_dsysv_rk_work*, *LAPACKE_ssysv_rk_work* and *LAPACKE_zsysv_rk_work*.

4.19.1695 LAPACKE_ssysv_rook

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssysv_rook(armpl_int_t matrix_layout, char uplo,
                               armpl_int_t n, armpl_int_t nrhs, float *a,
                               armpl_int_t lda, armpl_int_t *ipiv, float *b,
                               armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysv_rook*, *LAPACKE_dsysv_rook*, *LAPACKE_ssysv_rook* and *LAPACKE_zsysv_rook*. It also exists with a native Fortran interface as *ssysv_rook*.

4.19.1696 LAPACKE_ssysv_rook_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssysv_rook_work(armpl_int_t matrix_layout, char uplo,
                                    armpl_int_t n, armpl_int_t nrhs, float *a,
                                    armpl_int_t lda, armpl_int_t *ipiv,
                                    float *b, armpl_int_t ldb, float *work,
                                    armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysv_rook_work*, *LAPACKE_dsysv_rook_work*, *LAPACKE_ssysv_rook_work* and *LAPACKE_zsysv_rook_work*.

4.19.1697 LAPACKE_ssysv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssysv_work(armpl_int_t matrix_layout, char uplo,
                               armpl_int_t n, armpl_int_t nrhs, float *a,
                               armpl_int_t lda, armpl_int_t *ipiv, float *b,
                               armpl_int_t ldb, float *work,
                               armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysv_work*, *LAPACKE_dsysv_work*, *LAPACKE_ssysv_work* and *LAPACKE_zsysv_work*.

4.19.1698 LAPACKE_ssysvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssysvx(armpl_int_t matrix_layout, char fact, char uplo,
                           armpl_int_t n, armpl_int_t nrhs, const float *a,
                           armpl_int_t lda, float *af, armpl_int_t ldaf,
                           armpl_int_t *ipiv, const float *b, armpl_int_t ldb,
                           float *x, armpl_int_t ldx, float *rcond,
                           float *ferr, float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysvx*, *LAPACKE_dsysvx*, *LAPACKE_ssysvx* and *LAPACKE_zsysvx*. It also exists with a native Fortran interface as *ssysvx*.

4.19.1699 LAPACKE_ssysvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssysvx_work(armpl_int_t matrix_layout, char fact,
                                char uplo, armpl_int_t n, armpl_int_t nrhs,
                                const float *a, armpl_int_t lda, float *af,
                                armpl_int_t ldaf, armpl_int_t *ipiv,
                                const float *b, armpl_int_t ldb, float *x,
                                armpl_int_t ldx, float *rcond, float *ferr,
                                float *berr, float *work, armpl_int_t lwork,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysvx_work*, *LAPACKE_dsysvx_work*, *LAPACKE_ssysvx_work* and *LAPACKE_zsysvx_work*.

4.19.1700 LAPACKE_ssysvxx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssysvxx(armpl_int_t matrix_layout, char fact, char uplo,
                            armpl_int_t n, armpl_int_t nrhs, float *a,
                            armpl_int_t lda, float *af, armpl_int_t ldaf,
                            armpl_int_t *ipiv, char *equed, float *s,
                            float *b, armpl_int_t ldb, float *x,
                            armpl_int_t ldx, float *rcond, float *rpvgrw,
                            float *berr, armpl_int_t n_err_bnds,
                            float *err_bnds_norm, float *err_bnds_comp,
                            armpl_int_t nparams, float *params);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysvxx*, *LAPACKE_dsysvxx*, *LAPACKE_ssysvxx* and *LAPACKE_zsysvxx*. It also exists with a native Fortran interface as *ssysvxx*.

4.19.1701 LAPACKE_ssysvxx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssysvxx_work(armpl_int_t matrix_layout, char fact,
                                 char uplo, armpl_int_t n, armpl_int_t nrhs,
                                 float *a, armpl_int_t lda, float *af,
                                 armpl_int_t ldaf, armpl_int_t *ipiv,
                                 char *equed, float *s, float *b,
                                 armpl_int_t ldb, float *x, armpl_int_t ldx,
                                 float *rcond, float *rpvgrw, float *berr,
                                 armpl_int_t n_err_bnds, float *err_bnds_norm,
                                 float *err_bnds_comp, armpl_int_t nparams,
                                 float *params, float *work,
                                 armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysvxx_work*, *LAPACKE_dsysvxx_work*, *LAPACKE_ssysvxx_work* and *LAPACKE_zsysvxx_work*.

4.19.1702 LAPACKE_ssyswapr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssyswapr(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, float *a, armpl_int_t lda,
                             armpl_int_t i1, armpl_int_t i2);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyswapr*, *LAPACKE_dsyswapr*, *LAPACKE_ssyswapr* and *LAPACKE_zsyswapr*. It also exists with a native Fortran interface as *ssyswapr*.

4.19.1703 LAPACKE_ssyswapr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssyswapr_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, float *a, armpl_int_t lda,
                                  armpl_int_t i1, armpl_int_t i2);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyswapr_work*, *LAPACKE_dsyswapr_work*, *LAPACKE_ssyswapr_work* and *LAPACKE_zsyswapr_work*.

4.19.1704 LAPACKE_ssytrd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssytrd(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, float *a, armpl_int_t lda, float *d,
                           float *e, float *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsytrd* and *LAPACKE_ssytrd*. It also exists with a native Fortran interface as *ssytrd*.

4.19.1705 LAPACKE_ssytrd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssytrd_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, float *a, armpl_int_t lda,
                                float *d, float *e, float *tau, float *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_dsytrd_work* and *LAPACKE_ssytrd_work*.

4.19.1706 LAPACKE_ssytrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssytrf(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, float *a, armpl_int_t lda,
                           armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrf*, *LAPACKE_dsytrf*, *LAPACKE_ssytrf* and *LAPACKE_zsytrf*. It also exists with a native Fortran interface as *ssytrf*.

4.19.1707 LAPACKE_ssytrf_aa

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssytrf_aa(armpl_int_t matrix_layout, char uplo,
                              armpl_int_t n, float *a, armpl_int_t lda,
                              armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrf_aa*, *LAPACKE_dsytrf_aa*, *LAPACKE_ssytrf_aa* and *LAPACKE_zsytrf_aa*. It also exists with a native Fortran interface as *ssytrf_aa*.

4.19.1708 LAPACKE_ssytrf_aa_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssytrf_aa_2stage(armpl_int_t matrix_layout, char uplo,
                                     armpl_int_t n, float *a, armpl_int_t lda,
                                     float *tb, armpl_int_t ltb,
                                     armpl_int_t *ipiv, armpl_int_t *ipiv2);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrf_aa_2stage*, *LAPACKE_dsytrf_aa_2stage*, *LAPACKE_ssytrf_aa_2stage* and *LAPACKE_zsytrf_aa_2stage*. It also exists with a native Fortran interface as *ssytrf_aa_2stage*.

4.19.1709 LAPACKE_ssytrf_aa_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssytrf_aa_work(armpl_int_t matrix_layout, char uplo,
                                   armpl_int_t n, float *a, armpl_int_t lda,
                                   armpl_int_t *ipiv, float *work,
                                   armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrf_aa_work*, *LAPACKE_dsytrf_aa_work*, *LAPACKE_ssytrf_aa_work* and *LAPACKE_zsytrf_aa_work*.

4.19.1710 LAPACKE_ssytrf_rk

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssytrf_rk(armpl_int_t matrix_layout, char uplo,
                              armpl_int_t n, float *a, armpl_int_t lda,
                              float *e, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrf_rk*, *LAPACKE_dsytrf_rk*, *LAPACKE_ssytrf_rk* and *LAPACKE_zsytrf_rk*. It also exists with a native Fortran interface as *ssytrf_rk*.

4.19.1711 LAPACKE_ssytrf_rk_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssytrf_rk_work(armpl_int_t matrix_layout, char uplo,
                                   armpl_int_t n, float *a, armpl_int_t lda,
                                   float *e, armpl_int_t *ipiv, float *work,
                                   armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrf_rk_work*, *LAPACKE_dsytrf_rk_work*, *LAPACKE_ssytrf_rk_work* and *LAPACKE_zsytrf_rk_work*.

4.19.1712 LAPACKE_ssytrf_rook

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssytrf_rook(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, float *a, armpl_int_t lda,
                                armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrf_rook*, *LAPACKE_dsytrf_rook*, *LAPACKE_ssytrf_rook* and *LAPACKE_zsytrf_rook*. It also exists with a native Fortran interface as *ssytrf_rook*.

4.19.1713 LAPACKE_ssytrf_rook_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssytrf_rook_work(armpl_int_t matrix_layout, char uplo,
                                     armpl_int_t n, float *a, armpl_int_t lda,
                                     armpl_int_t *ipiv, float *work,
                                     armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrf_rook_work*, *LAPACKE_dsytrf_rook_work*, *LAPACKE_ssytrf_rook_work* and *LAPACKE_zsytrf_rook_work*.

4.19.1714 LAPACKE_ssytrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssytrf_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, float *a, armpl_int_t lda,
                                armpl_int_t *ipiv, float *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrf_work*, *LAPACKE_dsytrf_work*, *LAPACKE_ssytrf_work* and *LAPACKE_zsytrf_work*.

4.19.1715 LAPACKE_ssytri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssytri(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, float *a, armpl_int_t lda,
                           const armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytri*, *LAPACKE_dsytri*, *LAPACKE_ssytri* and *LAPACKE_zsytri*. It also exists with a native Fortran interface as *ssytri*.

4.19.1716 LAPACKE_ssytri2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssytri2(armpl_int_t matrix_layout, char uplo,
                            armpl_int_t n, float *a, armpl_int_t lda,
                            const armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytri2*, *LAPACKE_dsytri2*, *LAPACKE_ssytri2* and *LAPACKE_zsytri2*. It also exists with a native Fortran interface as *ssytri2*.

4.19.1717 LAPACKE_ssytri2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssytri2_work(armpl_int_t matrix_layout, char uplo,
                                 armpl_int_t n, float *a, armpl_int_t lda,
                                 const armpl_int_t *ipiv, float *work,
                                 armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytri2_work*, *LAPACKE_dsytri2_work*, *LAPACKE_ssytri2_work* and *LAPACKE_zsytri2_work*.

4.19.1718 LAPACKE_ssytri2x

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssytri2x(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, float *a, armpl_int_t lda,
                             const armpl_int_t *ipiv, armpl_int_t nb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytri2x*, *LAPACKE_dsytri2x*, *LAPACKE_ssytri2x* and *LAPACKE_zsytri2x*. It also exists with a native Fortran interface as *ssytri2x*.

4.19.1719 LAPACKE_ssytri2x_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssytri2x_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, float *a, armpl_int_t lda,
                                  const armpl_int_t *ipiv, float *work,
                                  armpl_int_t nb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytri2x_work*, *LAPACKE_dsytri2x_work*, *LAPACKE_ssytri2x_work* and *LAPACKE_zsytri2x_work*.

4.19.1720 LAPACKE_ssytri_3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssytri_3(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, float *a, armpl_int_t lda,
                             const float *e, const armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytri_3*, *LAPACKE_dsytri_3*, *LAPACKE_ssytri_3* and *LAPACKE_zsytri_3*. It also exists with a native Fortran interface as *ssytri_3*.

4.19.1721 LAPACKE_ssytri_3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssytri_3_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, float *a, armpl_int_t lda,
                                  const float *e, const armpl_int_t *ipiv,
                                  float *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytri_3_work*, *LAPACKE_dsytri_3_work*, *LAPACKE_ssytri_3_work* and *LAPACKE_zsytri_3_work*.

4.19.1722 LAPACKE_ssytri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssytri_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, float *a, armpl_int_t lda,
                                const armpl_int_t *ipiv, float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytri_work*, *LAPACKE_dsytri_work*, *LAPACKE_ssytri_work* and *LAPACKE_zsytri_work*.

4.19.1723 LAPACKE_ssytrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssytrs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs, const float *a,
                           armpl_int_t lda, const armpl_int_t *ipiv, float *b,
                           armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs*, *LAPACKE_dsytrs*, *LAPACKE_ssytrs* and *LAPACKE_zsytrs*. It also exists with a native Fortran interface as *ssytrs*.

4.19.1724 LAPACKE_ssytrs2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssytrs2(armpl_int_t matrix_layout, char uplo,
                            armpl_int_t n, armpl_int_t nrhs, const float *a,
                            armpl_int_t lda, const armpl_int_t *ipiv,
                            float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs2*, *LAPACKE_dsytrs2*, *LAPACKE_ssytrs2* and *LAPACKE_zsytrs2*. It also exists with a native Fortran interface as *ssytrs2*.

4.19.1725 LAPACKE_ssytrs2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssytrs2_work(armpl_int_t matrix_layout, char uplo,
                                 armpl_int_t n, armpl_int_t nrhs,
                                 const float *a, armpl_int_t lda,
                                 const armpl_int_t *ipiv, float *b,
                                 armpl_int_t ldb, float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs2_work*, *LAPACKE_dsytrs2_work*, *LAPACKE_ssytrs2_work* and *LAPACKE_zsytrs2_work*.

4.19.1726 LAPACKE_ssytrs_3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssytrs_3(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, armpl_int_t nrhs, const float *a,
                             armpl_int_t lda, const float *e,
                             const armpl_int_t *ipiv, float *b,
                             armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs_3*, *LAPACKE_dsytrs_3*, *LAPACKE_ssytrs_3* and *LAPACKE_zsytrs_3*. It also exists with a native Fortran interface as *ssytrs_3*.

4.19.1727 LAPACKE_ssytrs_3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssytrs_3_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, armpl_int_t nrhs,
                                  const float *a, armpl_int_t lda,
                                  const float *e, const armpl_int_t *ipiv,
                                  float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs_3_work*, *LAPACKE_dsytrs_3_work*, *LAPACKE_ssytrs_3_work* and *LAPACKE_zsytrs_3_work*.

4.19.1728 LAPACKE_ssytrs_aa

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssytrs_aa(armpl_int_t matrix_layout, char uplo,
                              armpl_int_t n, armpl_int_t nrhs, const float *a,
                              armpl_int_t lda, const armpl_int_t *ipiv,
                              float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs_aa*, *LAPACKE_dsytrs_aa*, *LAPACKE_ssytrs_aa* and *LAPACKE_zsytrs_aa*. It also exists with a native Fortran interface as *ssytrs_aa*.

4.19.1729 LAPACKE_ssytrs_aa_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssytrs_aa_2stage(armpl_int_t matrix_layout, char uplo,
                                     armpl_int_t n, armpl_int_t nrhs,
                                     float *a, armpl_int_t lda, float *tb,
                                     armpl_int_t ltb, armpl_int_t *ipiv,
                                     armpl_int_t *ipiv2, float *b,
                                     armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs_aa_2stage*, *LAPACKE_dsytrs_aa_2stage*, *LAPACKE_ssytrs_aa_2stage* and *LAPACKE_zsytrs_aa_2stage*. It also exists with a native Fortran interface as *ssytrs_aa_2stage*.

4.19.1730 LAPACKE_ssytrs_aa_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssytrs_aa_work(armpl_int_t matrix_layout, char uplo,
                                   armpl_int_t n, armpl_int_t nrhs,
                                   const float *a, armpl_int_t lda,
                                   const armpl_int_t *ipiv, float *b,
                                   armpl_int_t ldb, float *work,
                                   armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs_aa_work*, *LAPACKE_dsytrs_aa_work*, *LAPACKE_ssytrs_aa_work* and *LAPACKE_zsytrs_aa_work*.

4.19.1731 LAPACKE_ssytrs_rook

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssytrs_rook(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const float *a, armpl_int_t lda,
                                const armpl_int_t *ipiv, float *b,
                                armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.1732 LAPACKE_ssytrs_rook_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssytrs_rook_work(armpl_int_t matrix_layout, char uplo,
                                     armpl_int_t n, armpl_int_t nrhs,
                                     const float *a, armpl_int_t lda,
                                     const armpl_int_t *ipiv, float *b,
                                     armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs_rook_work*, *LAPACKE_dsytrs_rook_work*, *LAPACKE_ssytrs_rook_work* and *LAPACKE_zsytrs_rook_work*.

4.19.1733 LAPACKE_ssytrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ssytrs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const float *a, armpl_int_t lda,
                                const armpl_int_t *ipiv, float *b,
                                armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs_work*, *LAPACKE_dsytrs_work*, *LAPACKE_ssytrs_work* and *LAPACKE_zsytrs_work*.

4.19.1734 LAPACKE_stbcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stbcon(armpl_int_t matrix_layout, char norm, char uplo,
                           char diag, armpl_int_t n, armpl_int_t kd,
                           const float *ab, armpl_int_t ldab, float *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctbcon*, *LAPACKE_dtbcon*, *LAPACKE_stbcon* and *LAPACKE_ztbcon*. It also exists with a native Fortran interface as *stbcon*.

4.19.1735 LAPACKE_stbcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stbcon_work(armpl_int_t matrix_layout, char norm,
                                char uplo, char diag, armpl_int_t n,
                                armpl_int_t kd, const float *ab,
                                armpl_int_t ldab, float *rcond, float *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctbcon_work*, *LAPACKE_dtbcon_work*, *LAPACKE_stbcon_work* and *LAPACKE_ztbcon_work*.

4.19.1736 LAPACKE_stbrfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stbrfs(armpl_int_t matrix_layout, char uplo, char trans,
                           char diag, armpl_int_t n, armpl_int_t kd,
                           armpl_int_t nrhs, const float *ab,
                           armpl_int_t ldab, const float *b, armpl_int_t ldb,
                           const float *x, armpl_int_t ldx, float *ferr,
                           float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctbrfs*, *LAPACKE_dtbrfs*, *LAPACKE_stbrfs* and *LAPACKE_ztbrfs*. It also exists with a native Fortran interface as *stbrfs*.

4.19.1737 LAPACKE_stbrfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stbrfs_work(armpl_int_t matrix_layout, char uplo,
                                char trans, char diag, armpl_int_t n,
                                armpl_int_t kd, armpl_int_t nrhs,
                                const float *ab, armpl_int_t ldab,
                                const float *b, armpl_int_t ldb,
                                const float *x, armpl_int_t ldx, float *ferr,
                                float *berr, float *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctbrfs_work*, *LAPACKE_dtbrfs_work*, *LAPACKE_stbrfs_work* and *LAPACKE_ztbrfs_work*.

4.19.1738 LAPACKE_stbtrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stbtrs(armpl_int_t matrix_layout, char uplo, char trans,
                           char diag, armpl_int_t n, armpl_int_t kd,
                           armpl_int_t nrhs, const float *ab,
                           armpl_int_t ldab, float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctbtrs*, *LAPACKE_dtbtrs*, *LAPACKE_stbtrs* and *LAPACKE_ztbtrs*. It also exists with a native Fortran interface as *stbtrs*.

4.19.1739 LAPACKE_stbtrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stbtrs_work(armpl_int_t matrix_layout, char uplo,
                                char trans, char diag, armpl_int_t n,
                                armpl_int_t kd, armpl_int_t nrhs,
                                const float *ab, armpl_int_t ldab, float *b,
                                armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctbtrs_work*, *LAPACKE_dtbtrs_work*, *LAPACKE_stbtrs_work* and *LAPACKE_ztbtrs_work*.

4.19.1740 LAPACKE_stfsm

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stfsm(armpl_int_t matrix_layout, char transr, char side,
                          char uplo, char trans, char diag, armpl_int_t m,
                          armpl_int_t n, float alpha, const float *a,
                          float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctfsm*, *LAPACKE_dtfsm*, *LAPACKE_stfsm* and *LAPACKE_ztfsm*. It also exists with a native Fortran interface as *stfsm*.

4.19.1741 LAPACKE_stfsm_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stfsm_work(armpl_int_t matrix_layout, char transr,
                               char side, char uplo, char trans, char diag,
                               armpl_int_t m, armpl_int_t n, float alpha,
                               const float *a, float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctfsm_work*, *LAPACKE_dtfsm_work*, *LAPACKE_stfsm_work* and *LAPACKE_ztfsm_work*.

4.19.1742 LAPACKE_stftri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stftri(armpl_int_t matrix_layout, char transr, char uplo,
                           char diag, armpl_int_t n, float *a);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctftri*, *LAPACKE_dtftri*, *LAPACKE_stftri* and *LAPACKE_ztftri*. It also exists with a native Fortran interface as *stftri*.

4.19.1743 LAPACKE_stftri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stftri_work(armpl_int_t matrix_layout, char transr,
                                char uplo, char diag, armpl_int_t n,
                                float *a);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctftri_work*, *LAPACKE_dtftri_work*, *LAPACKE_stftri_work* and *LAPACKE_ztftri_work*.

4.19.1744 LAPACKE_stfttp

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stfttp(armpl_int_t matrix_layout, char transr, char uplo,
                           armpl_int_t n, const float *arf, float *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctfttp*, *LAPACKE_dtfttp*, *LAPACKE_stfttp* and *LAPACKE_ztfttp*. It also exists with a native Fortran interface as *stfttp*.

4.19.1745 LAPACKE_stfttp_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stfttp_work(armpl_int_t matrix_layout, char transr,
                                char uplo, armpl_int_t n, const float *arf,
                                float *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctfttp_work*, *LAPACKE_dtfttp_work*, *LAPACKE_stfttp_work* and *LAPACKE_ztfttp_work*.

4.19.1746 LAPACKE_stfttr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stfttr(armpl_int_t matrix_layout, char transr, char uplo,
                           armpl_int_t n, const float *arf, float *a,
                           armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctfttr*, *LAPACKE_dtfttr*, *LAPACKE_stfttr* and *LAPACKE_ztfttr*. It also exists with a native Fortran interface as *stfttr*.

4.19.1747 LAPACKE_stfttr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stfttr_work(armpl_int_t matrix_layout, char transr,
                                char uplo, armpl_int_t n, const float *arf,
                                float *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctfttr_work*, *LAPACKE_dtfttr_work*, *LAPACKE_stfttr_work* and *LAPACKE_ztfttr_work*.

4.19.1748 LAPACKE_stgevc

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stgevc(armpl_int_t matrix_layout, char side, char howmny,
                           const armpl_int_t *select, armpl_int_t n,
                           const float *s, armpl_int_t lds, const float *p,
                           armpl_int_t ldp, float *vl, armpl_int_t ldvl,
                           float *vr, armpl_int_t ldvr, armpl_int_t mm,
                           armpl_int_t *m);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgevc*, *LAPACKE_dtgevc*, *LAPACKE_stgevc* and *LAPACKE_ztgevc*. It also exists with a native Fortran interface as *stgevc*.

4.19.1749 LAPACKE_stgevc_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stgevc_work(armpl_int_t matrix_layout, char side,
                                char howmny, const armpl_int_t *select,
                                armpl_int_t n, const float *s,
```

(continues on next page)

(continued from previous page)

```
                                armpl_int_t lds, const float *p,
                                armpl_int_t ldp, float *vl, armpl_int_t ldvl,
                                float *vr, armpl_int_t ldvr, armpl_int_t mm,
                                armpl_int_t *m, float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgevc_work*, *LAPACKE_dtgevc_work*, *LAPACKE_stgevc_work* and *LAPACKE_ztgevc_work*.

4.19.1750 LAPACKE_stgexc

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stgexc(armpl_int_t matrix_layout, armpl_int_t wantq,
                           armpl_int_t wantz, armpl_int_t n, float *a,
                           armpl_int_t lda, float *b, armpl_int_t ldb,
                           float *q, armpl_int_t ldq, float *z,
                           armpl_int_t ldz, armpl_int_t *ifst,
                           armpl_int_t *ilst);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgexc*, *LAPACKE_dtgexc*, *LAPACKE_stgexc* and *LAPACKE_ztgexc*. It also exists with a native Fortran interface as *stgexc*.

4.19.1751 LAPACKE_stgexc_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stgexc_work(armpl_int_t matrix_layout, armpl_int_t wantq,
                                armpl_int_t wantz, armpl_int_t n, float *a,
                                armpl_int_t lda, float *b, armpl_int_t ldb,
                                float *q, armpl_int_t ldq, float *z,
                                armpl_int_t ldz, armpl_int_t *ifst,
                                armpl_int_t *ilst, float *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgexc_work*, *LAPACKE_dtgexc_work*, *LAPACKE_stgexc_work* and *LAPACKE_ztgexc_work*.

4.19.1752 LAPACKE_stgsen

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stgsen(armpl_int_t matrix_layout, armpl_int_t ijob,
                           armpl_int_t wantq, armpl_int_t wantz,
                           const armpl_int_t *select, armpl_int_t n, float *a,
                           armpl_int_t lda, float *b, armpl_int_t ldb,
                           float *alphar, float *alphai, float *beta,
                           float *q, armpl_int_t ldq, float *z,
                           armpl_int_t ldz, armpl_int_t *m, float *pl,
                           float *pr, float *dif);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgsen*, *LAPACKE_dtgsen*, *LAPACKE_stgsen* and *LAPACKE_ztgsen*. It also exists with a native Fortran interface as *stgsen*.

4.19.1753 LAPACKE_stgsen_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stgsen_work(armpl_int_t matrix_layout, armpl_int_t ijob,
                                armpl_int_t wantq, armpl_int_t wantz,
                                const armpl_int_t *select, armpl_int_t n,
                                float *a, armpl_int_t lda, float *b,
                                armpl_int_t ldb, float *alphar, float *alphai,
                                float *beta, float *q, armpl_int_t ldq,
                                float *z, armpl_int_t ldz, armpl_int_t *m,
                                float *pl, float *pr, float *dif, float *work,
                                armpl_int_t lwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgsen_work*, *LAPACKE_dtgsen_work*, *LAPACKE_stgsen_work* and *LAPACKE_ztgsen_work*.

4.19.1754 LAPACKE_stgsja

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stgsja(armpl_int_t matrix_layout, char jobu, char jobv,
                           char jobq, armpl_int_t m, armpl_int_t p,
                           armpl_int_t n, armpl_int_t k, armpl_int_t l,
                           float *a, armpl_int_t lda, float *b,
                           armpl_int_t ldb, float tola, float tolb,
                           float *alpha, float *beta, float *u,
                           armpl_int_t ldu, float *v, armpl_int_t ldv,
                           float *q, armpl_int_t ldq, armpl_int_t *ncycle);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.1755 LAPACKE_stgsja_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stgsja_work(armpl_int_t matrix_layout, char jobu,
                                char jobv, char jobq, armpl_int_t m,
                                armpl_int_t p, armpl_int_t n, armpl_int_t k,
                                armpl_int_t l, float *a, armpl_int_t lda,
                                float *b, armpl_int_t ldb, float tola,
                                float tolb, float *alpha, float *beta,
                                float *u, armpl_int_t ldu, float *v,
                                armpl_int_t ldv, float *q, armpl_int_t ldq,
                                float *work, armpl_int_t *ncycle);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgsja_work*, *LAPACKE_dtgsja_work*, *LAPACKE_stgsja_work* and *LAPACKE_ztgsja_work*.

4.19.1756 LAPACKE_stgsna

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stgsna(armpl_int_t matrix_layout, char job, char howmny,
                           const armpl_int_t *select, armpl_int_t n,
                           const float *a, armpl_int_t lda, const float *b,
                           armpl_int_t ldb, const float *vl, armpl_int_t ldvl,
```

(continues on next page)

(continued from previous page)

```
                                const float *vr, armpl_int_t ldvr, float *s,
                                float *dif, armpl_int_t mm, armpl_int_t *m);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgsna*, *LAPACKE_dtgsna*, *LAPACKE_stgsna* and *LAPACKE_ztgsna*. It also exists with a native Fortran interface as *stgsna*.

4.19.1757 LAPACKE_stgsna_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stgsna_work(armpl_int_t matrix_layout, char job,
                                char howmny, const armpl_int_t *select,
                                armpl_int_t n, const float *a,
                                armpl_int_t lda, const float *b,
                                armpl_int_t ldb, const float *vl,
                                armpl_int_t ldvl, const float *vr,
                                armpl_int_t ldvr, float *s, float *dif,
                                armpl_int_t mm, armpl_int_t *m, float *work,
                                armpl_int_t lwork, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgsna_work*, *LAPACKE_dtgsna_work*, *LAPACKE_stgsna_work* and *LAPACKE_ztgsna_work*.

4.19.1758 LAPACKE_stgsyl

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stgsyl(armpl_int_t matrix_layout, char trans,
                           armpl_int_t ijob, armpl_int_t m, armpl_int_t n,
                           const float *a, armpl_int_t lda, const float *b,
                           armpl_int_t ldb, float *c, armpl_int_t ldc,
                           const float *d, armpl_int_t ldd, const float *e,
                           armpl_int_t lde, float *f, armpl_int_t ldf,
                           float *scale, float *dif);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgsyl*, *LAPACKE_dtgsyl*, *LAPACKE_stgsyl* and *LAPACKE_ztgsyl*. It also exists with a native Fortran interface as *stgsyl*.

4.19.1759 LAPACKE_stgsyl_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stgsyl_work(armpl_int_t matrix_layout, char trans,
                                armpl_int_t ijob, armpl_int_t m,
                                armpl_int_t n, const float *a,
                                armpl_int_t lda, const float *b,
                                armpl_int_t ldb, float *c, armpl_int_t ldc,
                                const float *d, armpl_int_t ldd,
                                const float *e, armpl_int_t lde, float *f,
                                armpl_int_t ldf, float *scale, float *dif,
                                float *work, armpl_int_t lwork,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgsyl_work*, *LAPACKE_dtgsyl_work*, *LAPACKE_stgsyl_work* and *LAPACKE_ztgsyl_work*.

4.19.1760 LAPACKE_stpcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stpcon(armpl_int_t matrix_layout, char norm, char uplo,
                           char diag, armpl_int_t n, const float *ap,
                           float *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpcon*, *LAPACKE_dtpcon*, *LAPACKE_stpcon* and *LAPACKE_ztpcon*. It also exists with a native Fortran interface as *stpcon*.

4.19.1761 LAPACKE_stpcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stpcon_work(armpl_int_t matrix_layout, char norm,
                                char uplo, char diag, armpl_int_t n,
                                const float *ap, float *rcond, float *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpcon_work*, *LAPACKE_dtpcon_work*, *LAPACKE_stpcon_work* and *LAPACKE_ztpcon_work*.

4.19.1762 LAPACKE_stpmqrt

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stpmqrt(armpl_int_t matrix_layout, char side, char trans,
                            armpl_int_t m, armpl_int_t n, armpl_int_t k,
                            armpl_int_t l, armpl_int_t nb, const float *v,
                            armpl_int_t ldv, const float *t, armpl_int_t ldt,
                            float *a, armpl_int_t lda, float *b,
                            armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpmqrt*, *LAPACKE_dtpmqrt*, *LAPACKE_stpmqrt* and *LAPACKE_ztpmqrt*. It also exists with a native Fortran interface as *stpmqrt*.

4.19.1763 LAPACKE_stpmqrt_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stpmqrt_work(armpl_int_t matrix_layout, char side,
                                 char trans, armpl_int_t m, armpl_int_t n,
                                 armpl_int_t k, armpl_int_t l, armpl_int_t nb,
                                 const float *v, armpl_int_t ldv,
                                 const float *t, armpl_int_t ldt, float *a,
                                 armpl_int_t lda, float *b, armpl_int_t ldb,
                                 float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpmqrt_work*, *LAPACKE_dtpmqrt_work*, *LAPACKE_stpmqrt_work* and *LAPACKE_ztpmqrt_work*.

4.19.1764 LAPACKE_stpqrt

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stpqrt(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t l, armpl_int_t nb,
                           float *a, armpl_int_t lda, float *b,
                           armpl_int_t ldb, float *t, armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpqrt*, *LAPACKE_dtpqrt*, *LAPACKE_stpqrt* and *LAPACKE_ztpqrt*. It also exists with a native Fortran interface as *stpqrt*.

4.19.1765 LAPACKE_stpqrt2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stpqrt2(armpl_int_t matrix_layout, armpl_int_t m,
                            armpl_int_t n, armpl_int_t l, float *a,
                            armpl_int_t lda, float *b, armpl_int_t ldb,
                            float *t, armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpqrt2*, *LAPACKE_dtpqrt2*, *LAPACKE_stpqrt2* and *LAPACKE_ztpqrt2*. It also exists with a native Fortran interface as *stpqrt2*.

4.19.1766 LAPACKE_stpqrt2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stpqrt2_work(armpl_int_t matrix_layout, armpl_int_t m,
                                 armpl_int_t n, armpl_int_t l, float *a,
                                 armpl_int_t lda, float *b, armpl_int_t ldb,
                                 float *t, armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpqrt2_work*, *LAPACKE_dtpqrt2_work*, *LAPACKE_stpqrt2_work* and *LAPACKE_ztpqrt2_work*.

4.19.1767 LAPACKE_stpqrt_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stpqrt_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t l, armpl_int_t nb,
                                float *a, armpl_int_t lda, float *b,
                                armpl_int_t ldb, float *t, armpl_int_t ldt,
                                float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpqrt_work*, *LAPACKE_dtpqrt_work*, *LAPACKE_stpqrt_work* and *LAPACKE_ztpqrt_work*.

4.19.1768 LAPACKE_stprfb

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stprfb(armpl_int_t matrix_layout, char side, char trans,
                           char direct, char storev, armpl_int_t m,
                           armpl_int_t n, armpl_int_t k, armpl_int_t l,
                           const float *v, armpl_int_t ldv, const float *t,
                           armpl_int_t ldt, float *a, armpl_int_t lda,
                           float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctprfb*, *LAPACKE_dtprfb*, *LAPACKE_stprfb* and *LAPACKE_ztprfb*. It also exists with a native Fortran interface as *stprfb*.

4.19.1769 LAPACKE_stprfb_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stprfb_work(armpl_int_t matrix_layout, char side,
                                char trans, char direct, char storev,
                                armpl_int_t m, armpl_int_t n, armpl_int_t k,
```

(continues on next page)

(continued from previous page)

```
                                armpl_int_t l, const float *v,
                                armpl_int_t ldv, const float *t,
                                armpl_int_t ldt, float *a, armpl_int_t lda,
                                float *b, armpl_int_t ldb, float *work,
                                armpl_int_t ldwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctprfb_work*, *LAPACKE_dtprfb_work*, *LAPACKE_stprfb_work* and *LAPACKE_ztprfb_work*.

4.19.1770 LAPACKE_stprfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stprfs(armpl_int_t matrix_layout, char uplo, char trans,
                           char diag, armpl_int_t n, armpl_int_t nrhs,
                           const float *ap, const float *b, armpl_int_t ldb,
                           const float *x, armpl_int_t ldx, float *ferr,
                           float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctprfs*, *LAPACKE_dtprfs*, *LAPACKE_stprfs* and *LAPACKE_ztprfs*. It also exists with a native Fortran interface as *stprfs*.

4.19.1771 LAPACKE_stprfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stprfs_work(armpl_int_t matrix_layout, char uplo,
                                char trans, char diag, armpl_int_t n,
                                armpl_int_t nrhs, const float *ap,
                                const float *b, armpl_int_t ldb,
                                const float *x, armpl_int_t ldx, float *ferr,
                                float *berr, float *work,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctprfs_work*, *LAPACKE_dtprfs_work*, *LAPACKE_stprfs_work* and *LAPACKE_ztprfs_work*.

4.19.1772 LAPACKE_stptri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stptri(armpl_int_t matrix_layout, char uplo, char diag,
                           armpl_int_t n, float *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctptri*, *LAPACKE_dtptri*, *LAPACKE_stptri* and *LAPACKE_ztptri*. It also exists with a native Fortran interface as *stptri*.

4.19.1773 LAPACKE_stptri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stptri_work(armpl_int_t matrix_layout, char uplo,
                                char diag, armpl_int_t n, float *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctptri_work*, *LAPACKE_dtptri_work*, *LAPACKE_stptri_work* and *LAPACKE_ztptri_work*.

4.19.1774 LAPACKE_stptrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stptrs(armpl_int_t matrix_layout, char uplo, char trans,
                           char diag, armpl_int_t n, armpl_int_t nrhs,
                           const float *ap, float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctptrs*, *LAPACKE_dtptrs*, *LAPACKE_stptrs* and *LAPACKE_ztptrs*. It also exists with a native Fortran interface as *stptrs*.

4.19.1775 LAPACKE_stptrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stptrs_work(armpl_int_t matrix_layout, char uplo,
                                char trans, char diag, armpl_int_t n,
                                armpl_int_t nrhs, const float *ap, float *b,
                                armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctptrs_work*, *LAPACKE_dtptrs_work*, *LAPACKE_stptrs_work* and *LAPACKE_ztptrs_work*.

4.19.1776 LAPACKE_stpttf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stpttf(armpl_int_t matrix_layout, char transr, char uplo,
                           armpl_int_t n, const float *ap, float *arf);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpttf*, *LAPACKE_dtpttf*, *LAPACKE_stpttf* and *LAPACKE_ztpttf*. It also exists with a native Fortran interface as *stpttf*.

4.19.1777 LAPACKE_stpttf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stpttf_work(armpl_int_t matrix_layout, char transr,
                                char uplo, armpl_int_t n, const float *ap,
                                float *arf);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpttf_work*, *LAPACKE_dtpttf_work*, *LAPACKE_stpttf_work* and *LAPACKE_ztpttf_work*.

4.19.1778 LAPACKE_stpttr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stpttr(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, const float *ap, float *a,
                           armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpttr*, *LAPACKE_dtpttr*, *LAPACKE_stpttr* and *LAPACKE_ztpttr*. It also exists with a native Fortran interface as *stpttr*.

4.19.1779 LAPACKE_stpttr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stpttr_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, const float *ap, float *a,
                                armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpttr_work*, *LAPACKE_dtpttr_work*, *LAPACKE_stpttr_work* and *LAPACKE_ztpttr_work*.

4.19.1780 LAPACKE_strcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_strcon(armpl_int_t matrix_layout, char norm, char uplo,
                           char diag, armpl_int_t n, const float *a,
                           armpl_int_t lda, float *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrcon*, *LAPACKE_dtrcon*, *LAPACKE_strcon* and *LAPACKE_ztrcon*. It also exists with a native Fortran interface as *strcon*.

4.19.1781 LAPACKE_strcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_strcon_work(armpl_int_t matrix_layout, char norm,
                                char uplo, char diag, armpl_int_t n,
                                const float *a, armpl_int_t lda, float *rcond,
                                float *work, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrcon_work*, *LAPACKE_dtrcon_work*, *LAPACKE_strcon_work* and *LAPACKE_ztrcon_work*.

4.19.1782 LAPACKE_strevc

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_strevc(armpl_int_t matrix_layout, char side, char howmny,
                           armpl_int_t *select, armpl_int_t n, const float *t,
                           armpl_int_t ldt, float *vl, armpl_int_t ldvl,
                           float *vr, armpl_int_t ldvr, armpl_int_t mm,
                           armpl_int_t *m);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrevc*, *LAPACKE_dtrevc*, *LAPACKE_strevc* and *LAPACKE_ztrevc*. It also exists with a native Fortran interface as *strevc*.

4.19.1783 LAPACKE_strevc_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_strevc_work(armpl_int_t matrix_layout, char side,
                                char howmny, armpl_int_t *select,
                                armpl_int_t n, const float *t,
                                armpl_int_t ldt, float *vl, armpl_int_t ldvl,
                                float *vr, armpl_int_t ldvr, armpl_int_t mm,
                                armpl_int_t *m, float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrevc_work*, *LAPACKE_dtrevc_work*, *LAPACKE_strevc_work* and *LAPACKE_ztrevc_work*.

4.19.1784 LAPACKE_strexc

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_strexc(armpl_int_t matrix_layout, char compq,
                           armpl_int_t n, float *t, armpl_int_t ldt, float *q,
                           armpl_int_t ldq, armpl_int_t *ifst,
                           armpl_int_t *ilst);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrexc*, *LAPACKE_dtrexc*, *LAPACKE_strexc* and *LAPACKE_ztrexc*. It also exists with a native Fortran interface as *strexc*.

4.19.1785 LAPACKE_strexc_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_strexc_work(armpl_int_t matrix_layout, char compq,
                                armpl_int_t n, float *t, armpl_int_t ldt,
                                float *q, armpl_int_t ldq, armpl_int_t *ifst,
                                armpl_int_t *ilst, float *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrexc_work*, *LAPACKE_dtrexc_work*, *LAPACKE_strexc_work* and *LAPACKE_ztrexc_work*.

4.19.1786 LAPACKE_strrfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_strrfs(armpl_int_t matrix_layout, char uplo, char trans,
                           char diag, armpl_int_t n, armpl_int_t nrhs,
                           const float *a, armpl_int_t lda, const float *b,
                           armpl_int_t ldb, const float *x, armpl_int_t ldx,
                           float *ferr, float *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrrfs*, *LAPACKE_dtrrfs*, *LAPACKE_strrfs* and *LAPACKE_ztrrfs*. It also exists with a native Fortran interface as *strrfs*.

4.19.1787 LAPACKE_strrfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_strrfs_work(armpl_int_t matrix_layout, char uplo,
                                char trans, char diag, armpl_int_t n,
                                armpl_int_t nrhs, const float *a,
                                armpl_int_t lda, const float *b,
                                armpl_int_t ldb, const float *x,
                                armpl_int_t ldx, float *ferr, float *berr,
                                float *work, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrrfs_work*, *LAPACKE_dtrrfs_work*, *LAPACKE_strrfs_work* and *LAPACKE_ztrrfs_work*.

4.19.1788 LAPACKE_strsen

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_strsen(armpl_int_t matrix_layout, char job, char compq,
                           const armpl_int_t *select, armpl_int_t n, float *t,
                           armpl_int_t ldt, float *q, armpl_int_t ldq,
                           float *wr, float *wi, armpl_int_t *m, float *s,
                           float *sep);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrsen*, *LAPACKE_dtrsen*, *LAPACKE_strsen* and *LAPACKE_ztrsen*. It also exists with a native Fortran interface as *strsen*.

4.19.1789 LAPACKE_strsen_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_strsen_work(armpl_int_t matrix_layout, char job,
                                char compq, const armpl_int_t *select,
                                armpl_int_t n, float *t, armpl_int_t ldt,
                                float *q, armpl_int_t ldq, float *wr,
                                float *wi, armpl_int_t *m, float *s,
                                float *sep, float *work, armpl_int_t lwork,
                                armpl_int_t *iwork, armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrsen_work*, *LAPACKE_dtrsen_work*, *LAPACKE_strsen_work* and *LAPACKE_ztrsen_work*.

4.19.1790 LAPACKE_strsna

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_strsna(armpl_int_t matrix_layout, char job, char howmny,
                           const armpl_int_t *select, armpl_int_t n,
                           const float *t, armpl_int_t ldt, const float *vl,
                           armpl_int_t ldvl, const float *vr,
                           armpl_int_t ldvr, float *s, float *sep,
                           armpl_int_t mm, armpl_int_t *m);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrsna*, *LAPACKE_dtrsna*, *LAPACKE_strsna* and *LAPACKE_ztrsna*. It also exists with a native Fortran interface as *strsna*.

4.19.1791 LAPACKE_strsna_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_strsna_work(armpl_int_t matrix_layout, char job,
                                char howmny, const armpl_int_t *select,
                                armpl_int_t n, const float *t,
                                armpl_int_t ldt, const float *vl,
                                armpl_int_t ldvl, const float *vr,
                                armpl_int_t ldvr, float *s, float *sep,
                                armpl_int_t mm, armpl_int_t *m, float *work,
                                armpl_int_t ldwork, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrsna_work*, *LAPACKE_dtrsna_work*, *LAPACKE_strsna_work* and *LAPACKE_ztrsna_work*.

4.19.1792 LAPACKE_strsyl

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_strsyl(armpl_int_t matrix_layout, char trana, char tranb,
                           armpl_int_t isgn, armpl_int_t m, armpl_int_t n,
                           const float *a, armpl_int_t lda, const float *b,
                           armpl_int_t ldb, float *c, armpl_int_t ldc,
                           float *scale);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrsyl*, *LAPACKE_dtrsyl*, *LAPACKE_strsyl* and *LAPACKE_ztrsyl*. It also exists with a native Fortran interface as *strsyl*.

4.19.1793 LAPACKE_strsyl_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_strsyl_work(armpl_int_t matrix_layout, char trana,
                                char tranb, armpl_int_t isgn, armpl_int_t m,
                                armpl_int_t n, const float *a,
                                armpl_int_t lda, const float *b,
                                armpl_int_t ldb, float *c, armpl_int_t ldc,
                                float *scale);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrsyl_work*, *LAPACKE_dtrsyl_work*, *LAPACKE_strsyl_work* and *LAPACKE_ztrsyl_work*.

4.19.1794 LAPACKE_strtri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_strtri(armpl_int_t matrix_layout, char uplo, char diag,
                           armpl_int_t n, float *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrtri*, *LAPACKE_dtrtri*, *LAPACKE_strtri* and *LAPACKE_ztrtri*. It also exists with a native Fortran interface as *strtri*.

4.19.1795 LAPACKE_strtri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_strtri_work(armpl_int_t matrix_layout, char uplo,
                                char diag, armpl_int_t n, float *a,
                                armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrtri_work*, *LAPACKE_dtrtri_work*, *LAPACKE_strtri_work* and *LAPACKE_ztrtri_work*.

4.19.1796 LAPACKE_strtrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_strtrs(armpl_int_t matrix_layout, char uplo, char trans,
                           char diag, armpl_int_t n, armpl_int_t nrhs,
                           const float *a, armpl_int_t lda, float *b,
                           armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrtrs*, *LAPACKE_dtrtrs*, *LAPACKE_strtrs* and *LAPACKE_ztrtrs*. It also exists with a native Fortran interface as *strtrs*.

4.19.1797 LAPACKE_strtrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_strtrs_work(armpl_int_t matrix_layout, char uplo,
                                char trans, char diag, armpl_int_t n,
                                armpl_int_t nrhs, const float *a,
                                armpl_int_t lda, float *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrtrs_work*, *LAPACKE_dtrtrs_work*, *LAPACKE_strtrs_work* and *LAPACKE_ztrtrs_work*.

4.19.1798 LAPACKE_strttf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_strttf(armpl_int_t matrix_layout, char transr, char uplo,
                           armpl_int_t n, const float *a, armpl_int_t lda,
                           float *arf);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrttf*, *LAPACKE_dtrttf*, *LAPACKE_strttf* and *LAPACKE_ztrttf*. It also exists with a native Fortran interface as *strttf*.

4.19.1799 LAPACKE_strttf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_strttf_work(armpl_int_t matrix_layout, char transr,
                                char uplo, armpl_int_t n, const float *a,
                                armpl_int_t lda, float *arf);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrttf_work*, *LAPACKE_dtrttf_work*, *LAPACKE_strttf_work* and *LAPACKE_ztrttf_work*.

4.19.1800 LAPACKE_strttp

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_strttp(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, const float *a, armpl_int_t lda,
                           float *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrttp*, *LAPACKE_dtrttp*, *LAPACKE_strttp* and *LAPACKE_ztrttp*. It also exists with a native Fortran interface as *strttp*.

4.19.1801 LAPACKE_strttp_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_strttp_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, const float *a,
                                armpl_int_t lda, float *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrttp_work*, *LAPACKE_dtrttp_work*, *LAPACKE_strttp_work* and *LAPACKE_ztrttp_work*.

4.19.1802 LAPACKE_stzrzf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stzrzf(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, float *a, armpl_int_t lda,
                           float *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctzrzf*, *LAPACKE_dtzrzf*, *LAPACKE_stzrzf* and *LAPACKE_ztzrzf*. It also exists with a native Fortran interface as *stzrzf*.

4.19.1803 LAPACKE_stzrzf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_stzrzf_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, float *a, armpl_int_t lda,
                                float *tau, float *work, armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctzrzf_work*, *LAPACKE_dtzrzf_work*, *LAPACKE_stzrzf_work* and *LAPACKE_ztzrzf_work*.

4.19.1804 LAPACKE_zbbcsd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zbbcsd(armpl_int_t matrix_layout, char jobu1, char jobu2,
                           char jobv1t, char jobv2t, char trans,
                           armpl_int_t m, armpl_int_t p, armpl_int_t q,
                           double *theta, double *phi,
                           armpl_doublecomplex_t *u1, armpl_int_t ldu1,
                           armpl_doublecomplex_t *u2, armpl_int_t ldu2,
```

(continues on next page)

(continued from previous page)

```
                                armpl_doublecomplex_t *v1t, armpl_int_t ldv1t,
                                armpl_doublecomplex_t *v2t, armpl_int_t ldv2t,
                                double *b11d, double *b11e, double *b12d,
                                double *b12e, double *b21d, double *b21e,
                                double *b22d, double *b22e);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cbbcsd*, *LAPACKE_dbbcsd*, *LAPACKE_sbbcsd* and *LAPACKE_zbbcsd*. It also exists with a native Fortran interface as *zbbcsd*.

4.19.1805 LAPACKE_zbbcsd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zbbcsd_work(armpl_int_t matrix_layout, char jobu1,
                                char jobu2, char jobv1t, char jobv2t,
                                char trans, armpl_int_t m, armpl_int_t p,
                                armpl_int_t q, double *theta, double *phi,
                                armpl_doublecomplex_t *u1, armpl_int_t ldu1,
                                armpl_doublecomplex_t *u2, armpl_int_t ldu2,
                                armpl_doublecomplex_t *v1t, armpl_int_t ldv1t,
                                armpl_doublecomplex_t *v2t, armpl_int_t ldv2t,
                                double *b11d, double *b11e, double *b12d,
                                double *b12e, double *b21d, double *b21e,
                                double *b22d, double *b22e, double *rwork,
                                armpl_int_t lrwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cbbcsd_work*, *LAPACKE_dbbcsd_work*, *LAPACKE_sbbcsd_work* and *LAPACKE_zbbcsd_work*.

4.19.1806 LAPACKE_zbdsqr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zbdsqr(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t ncvt, armpl_int_t nru,
                           armpl_int_t ncc, double *d, double *e,
                           armpl_doublecomplex_t *vt, armpl_int_t ldvt,
                           armpl_doublecomplex_t *u, armpl_int_t ldu,
                           armpl_doublecomplex_t *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cbdsqr*, *LAPACKE_dbdsqr*, *LAPACKE_sbdsqr* and *LAPACKE_zbdsqr*. It also exists with a native Fortran interface as *zbdsqr*.

4.19.1807 LAPACKE_zbdsqr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zbdsqr_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t ncvt,
                                armpl_int_t nru, armpl_int_t ncc, double *d,
                                double *e, armpl_doublecomplex_t *vt,
                                armpl_int_t ldvt, armpl_doublecomplex_t *u,
                                armpl_int_t ldu, armpl_doublecomplex_t *c,
                                armpl_int_t ldc, double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.1808 LAPACKE_zcgesv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zcgesv(armpl_int_t matrix_layout, armpl_int_t n,
                           armpl_int_t nrhs, armpl_doublecomplex_t *a,
                           armpl_int_t lda, armpl_int_t *ipiv,
                           armpl_doublecomplex_t *b, armpl_int_t ldb,
                           armpl_doublecomplex_t *x, armpl_int_t ldx,
                           armpl_int_t *iter);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_zcgesv*. It also exists with a native Fortran interface as *zcgesv*.

4.19.1809 LAPACKE_zcgesv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zcgesv_work(armpl_int_t matrix_layout, armpl_int_t n,
                                armpl_int_t nrhs, armpl_doublecomplex_t *a,
                                armpl_int_t lda, armpl_int_t *ipiv,
                                armpl_doublecomplex_t *b, armpl_int_t ldb,
                                armpl_doublecomplex_t *x, armpl_int_t ldx,
                                armpl_doublecomplex_t *work,
                                armpl_singlecomplex_t *swork, double *rwork,
                                armpl_int_t *iter);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_zcgesv_work*.

4.19.1810 LAPACKE_zcposv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zcposv(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           armpl_doublecomplex_t *b, armpl_int_t ldb,
                           armpl_doublecomplex_t *x, armpl_int_t ldx,
                           armpl_int_t *iter);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_zcposv*. It also exists with a native Fortran interface as *zcposv*.

4.19.1811 LAPACKE_zcposv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zcposv_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                armpl_doublecomplex_t *a, armpl_int_t lda,
                                armpl_doublecomplex_t *b, armpl_int_t ldb,
                                armpl_doublecomplex_t *x, armpl_int_t ldx,
                                armpl_doublecomplex_t *work,
                                armpl_singlecomplex_t *swork, double *rwork,
                                armpl_int_t *iter);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_zcposv_work*.

4.19.1812 LAPACKE_zgbbrd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgbbrd(armpl_int_t matrix_layout, char vect,
                           armpl_int_t m, armpl_int_t n, armpl_int_t ncc,
                           armpl_int_t kl, armpl_int_t ku,
                           armpl_doublecomplex_t *ab, armpl_int_t ldab,
                           double *d, double *e, armpl_doublecomplex_t *q,
                           armpl_int_t ldq, armpl_doublecomplex_t *pt,
                           armpl_int_t ldpt, armpl_doublecomplex_t *c,
                           armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbbrd*, *LAPACKE_dgbbrd*, *LAPACKE_sgbbrd* and *LAPACKE_zgbbrd*. It also exists with a native Fortran interface as *zgbbrd*.

4.19.1813 LAPACKE_zgbbrd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgbbrd_work(armpl_int_t matrix_layout, char vect,
                                armpl_int_t m, armpl_int_t n, armpl_int_t ncc,
```

(continues on next page)

(continued from previous page)

```
                                armpl_int_t kl, armpl_int_t ku,
                                armpl_doublecomplex_t *ab, armpl_int_t ldab,
                                double *d, double *e,
                                armpl_doublecomplex_t *q, armpl_int_t ldq,
                                armpl_doublecomplex_t *pt, armpl_int_t ldpt,
                                armpl_doublecomplex_t *c, armpl_int_t ldc,
                                armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbbrd_work*, *LAPACKE_dgbbrd_work*, *LAPACKE_sgbbrd_work* and *LAPACKE_zgbbrd_work*.

4.19.1814 LAPACKE_zgbcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgbcon(armpl_int_t matrix_layout, char norm,
                           armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                           const armpl_doublecomplex_t *ab, armpl_int_t ldab,
                           const armpl_int_t *ipiv, double anorm,
                           double *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbcon*, *LAPACKE_dgbcon*, *LAPACKE_sgbcon* and *LAPACKE_zgbcon*. It also exists with a native Fortran interface as *zgbcon*.

4.19.1815 LAPACKE_zgbcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgbcon_work(armpl_int_t matrix_layout, char norm,
                                armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                                const armpl_doublecomplex_t *ab,
                                armpl_int_t ldab, const armpl_int_t *ipiv,
                                double anorm, double *rcond,
                                armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbcon_work*, *LAPACKE_dgbcon_work*, *LAPACKE_sgbcon_work* and *LAPACKE_zgbcon_work*.

4.19.1816 LAPACKE_zgbequ

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgbequ(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                           const armpl_doublecomplex_t *ab, armpl_int_t ldab,
                           double *r, double *c, double *rowcnd,
                           double *colcnd, double *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbequ*, *LAPACKE_dgbequ*, *LAPACKE_sgbequ* and *LAPACKE_zgbequ*. It also exists with a native Fortran interface as *zgbequ*.

4.19.1817 LAPACKE_zgbequ_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgbequ_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                                const armpl_doublecomplex_t *ab,
                                armpl_int_t ldab, double *r, double *c,
                                double *rowcnd, double *colcnd,
                                double *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbequ_work*, *LAPACKE_dgbequ_work*, *LAPACKE_sgbequ_work* and *LAPACKE_zgbequ_work*.

4.19.1818 LAPACKE_zgbequb

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgbequb(armpl_int_t matrix_layout, armpl_int_t m,
                            armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                            const armpl_doublecomplex_t *ab, armpl_int_t ldab,
                            double *r, double *c, double *rowcnd,
                            double *colcnd, double *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbequb*, *LAPACKE_dgbequb*, *LAPACKE_sgbequb* and *LAPACKE_zgbequb*. It also exists with a native Fortran interface as *zgbequb*.

4.19.1819 LAPACKE_zgbequb_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgbequb_work(armpl_int_t matrix_layout, armpl_int_t m,
                                 armpl_int_t n, armpl_int_t kl,
                                 armpl_int_t ku,
                                 const armpl_doublecomplex_t *ab,
                                 armpl_int_t ldab, double *r, double *c,
                                 double *rowcnd, double *colcnd,
                                 double *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbequb_work*, *LAPACKE_dgbequb_work*, *LAPACKE_sgbequb_work* and *LAPACKE_zgbequb_work*.

4.19.1820 LAPACKE_zgbrfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgbrfs(armpl_int_t matrix_layout, char trans,
                           armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                           armpl_int_t nrhs, const armpl_doublecomplex_t *ab,
                           armpl_int_t ldab, const armpl_doublecomplex_t *afb,
                           armpl_int_t ldafb, const armpl_int_t *ipiv,
                           const armpl_doublecomplex_t *b, armpl_int_t ldb,
                           armpl_doublecomplex_t *x, armpl_int_t ldx,
                           double *ferr, double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbrfs*, *LAPACKE_dgbrfs*, *LAPACKE_sgbrfs* and *LAPACKE_zgbrfs*. It also exists with a native Fortran interface as *zgbrfs*.

4.19.1821 LAPACKE_zgbrfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgbrfs_work(armpl_int_t matrix_layout, char trans,
                                armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                                armpl_int_t nrhs,
                                const armpl_doublecomplex_t *ab,
                                armpl_int_t ldab,
                                const armpl_doublecomplex_t *afb,
                                armpl_int_t ldafb, const armpl_int_t *ipiv,
                                const armpl_doublecomplex_t *b,
                                armpl_int_t ldb, armpl_doublecomplex_t *x,
                                armpl_int_t ldx, double *ferr, double *berr,
                                armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbrfs_work*, *LAPACKE_dgbrfs_work*, *LAPACKE_sgbrfs_work* and *LAPACKE_zgbrfs_work*.

4.19.1822 LAPACKE_zgbrfsx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgbrfsx(armpl_int_t matrix_layout, char trans, char equed,
                            armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                            armpl_int_t nrhs, const armpl_doublecomplex_t *ab,
                            armpl_int_t ldab,
                            const armpl_doublecomplex_t *afb,
                            armpl_int_t ldafb, const armpl_int_t *ipiv,
                            const double *r, const double *c,
                            const armpl_doublecomplex_t *b, armpl_int_t ldb,
                            armpl_doublecomplex_t *x, armpl_int_t ldx,
                            double *rcond, double *berr,
                            armpl_int_t n_err_bnds, double *err_bnds_norm,
                            double *err_bnds_comp, armpl_int_t nparams,
                            double *params);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbrfsx*, *LAPACKE_dgbrfsx*, *LAPACKE_sgbrfsx* and *LAPACKE_zgbrfsx*. It also exists with a native Fortran interface as *zgbrfsx*.

4.19.1823 LAPACKE_zgbrfsx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgbrfsx_work(armpl_int_t matrix_layout, char trans,
                                 char equed, armpl_int_t n, armpl_int_t kl,
                                 armpl_int_t ku, armpl_int_t nrhs,
                                 const armpl_doublecomplex_t *ab,
                                 armpl_int_t ldab,
                                 const armpl_doublecomplex_t *afb,
                                 armpl_int_t ldafb, const armpl_int_t *ipiv,
                                 const double *r, const double *c,
                                 const armpl_doublecomplex_t *b,
                                 armpl_int_t ldb, armpl_doublecomplex_t *x,
                                 armpl_int_t ldx, double *rcond, double *berr,
                                 armpl_int_t n_err_bnds,
                                 double *err_bnds_norm, double *err_bnds_comp,
                                 armpl_int_t nparams, double *params,
                                 armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbrfsx_work*, *LAPACKE_dgbrfsx_work*, *LAPACKE_sgbrfsx_work* and *LAPACKE_zgbrfsx_work*.

4.19.1824 LAPACKE_zgbsv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgbsv(armpl_int_t matrix_layout, armpl_int_t n,
                          armpl_int_t kl, armpl_int_t ku, armpl_int_t nrhs,
                          armpl_doublecomplex_t *ab, armpl_int_t ldab,
                          armpl_int_t *ipiv, armpl_doublecomplex_t *b,
                          armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbsv*, *LAPACKE_dgbsv*, *LAPACKE_sgbsv* and *LAPACKE_zgbsv*. It also exists with a native Fortran interface as *zgbsv*.

4.19.1825 LAPACKE_zgbsv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgbsv_work(armpl_int_t matrix_layout, armpl_int_t n,
                               armpl_int_t kl, armpl_int_t ku,
                               armpl_int_t nrhs, armpl_doublecomplex_t *ab,
                               armpl_int_t ldab, armpl_int_t *ipiv,
                               armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbsv_work*, *LAPACKE_dgbsv_work*, *LAPACKE_sgbsv_work* and *LAPACKE_zgbsv_work*.

4.19.1826 LAPACKE_zgbsvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgbsvx(armpl_int_t matrix_layout, char fact, char trans,
                           armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                           armpl_int_t nrhs, armpl_doublecomplex_t *ab,
                           armpl_int_t ldab, armpl_doublecomplex_t *afb,
                           armpl_int_t ldafb, armpl_int_t *ipiv, char *equed,
                           double *r, double *c, armpl_doublecomplex_t *b,
                           armpl_int_t ldb, armpl_doublecomplex_t *x,
                           armpl_int_t ldx, double *rcond, double *ferr,
                           double *berr, double *rpivot);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbsvx*, *LAPACKE_dgbsvx*, *LAPACKE_sgbsvx* and *LAPACKE_zgbsvx*. It also exists with a native Fortran interface as *zgbsvx*.

4.19.1827 LAPACKE_zgbsvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgbsvx_work(armpl_int_t matrix_layout, char fact,
                                char trans, armpl_int_t n, armpl_int_t kl,
                                armpl_int_t ku, armpl_int_t nrhs,
                                armpl_doublecomplex_t *ab, armpl_int_t ldab,
                                armpl_doublecomplex_t *afb, armpl_int_t ldafb,
                                armpl_int_t *ipiv, char *equed, double *r,
                                double *c, armpl_doublecomplex_t *b,
                                armpl_int_t ldb, armpl_doublecomplex_t *x,
                                armpl_int_t ldx, double *rcond, double *ferr,
                                double *berr, armpl_doublecomplex_t *work,
                                double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbsvx_work*, *LAPACKE_dgbsvx_work*, *LAPACKE_sgbsvx_work* and *LAPACKE_zgbsvx_work*.

4.19.1828 LAPACKE_zgbsvxx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgbsvxx(armpl_int_t matrix_layout, char fact, char trans,
                            armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                            armpl_int_t nrhs, armpl_doublecomplex_t *ab,
                            armpl_int_t ldab, armpl_doublecomplex_t *afb,
                            armpl_int_t ldafb, armpl_int_t *ipiv, char *equed,
                            double *r, double *c, armpl_doublecomplex_t *b,
                            armpl_int_t ldb, armpl_doublecomplex_t *x,
                            armpl_int_t ldx, double *rcond, double *rpvgrw,
                            double *berr, armpl_int_t n_err_bnds,
                            double *err_bnds_norm, double *err_bnds_comp,
                            armpl_int_t nparams, double *params);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbsvxx*, *LAPACKE_dgbsvxx*, *LAPACKE_sgbsvxx* and *LAPACKE_zgbsvxx*. It also exists with a native Fortran interface as *zgbsvxx*.

4.19.1829 LAPACKE_zgbsvxx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgbsvxx_work(armpl_int_t matrix_layout, char fact,
                                 char trans, armpl_int_t n, armpl_int_t kl,
                                 armpl_int_t ku, armpl_int_t nrhs,
                                 armpl_doublecomplex_t *ab, armpl_int_t ldab,
                                 armpl_doublecomplex_t *afb,
                                 armpl_int_t ldafb, armpl_int_t *ipiv,
                                 char *equed, double *r, double *c,
                                 armpl_doublecomplex_t *b, armpl_int_t ldb,
                                 armpl_doublecomplex_t *x, armpl_int_t ldx,
                                 double *rcond, double *rpvgrw, double *berr,
                                 armpl_int_t n_err_bnds,
                                 double *err_bnds_norm, double *err_bnds_comp,
                                 armpl_int_t nparams, double *params,
                                 armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbsvxx_work*, *LAPACKE_dgbsvxx_work*, *LAPACKE_sgbsvxx_work* and *LAPACKE_zgbsvxx_work*.

4.19.1830 LAPACKE_zgbtrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgbtrf(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
```

(continues on next page)

(continued from previous page)

```
                                armpl_doublecomplex_t *ab, armpl_int_t ldab,
                                armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbtrf*, *LAPACKE_dgbtrf*, *LAPACKE_sgbtrf* and *LAPACKE_zgbtrf*. It also exists with a native Fortran interface as *zgbtrf*.

4.19.1831 LAPACKE_zgbtrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgbtrf_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                                armpl_doublecomplex_t *ab, armpl_int_t ldab,
                                armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbtrf_work*, *LAPACKE_dgbtrf_work*, *LAPACKE_sgbtrf_work* and *LAPACKE_zgbtrf_work*.

4.19.1832 LAPACKE_zgbtrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgbtrs(armpl_int_t matrix_layout, char trans,
                           armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                           armpl_int_t nrhs, const armpl_doublecomplex_t *ab,
                           armpl_int_t ldab, const armpl_int_t *ipiv,
                           armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbtrs*, *LAPACKE_dgbtrs*, *LAPACKE_sgbtrs* and *LAPACKE_zgbtrs*. It also exists with a native Fortran interface as *zgbtrs*.

4.19.1833 LAPACKE_zgbtrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgbtrs_work(armpl_int_t matrix_layout, char trans,
                                armpl_int_t n, armpl_int_t kl, armpl_int_t ku,
                                armpl_int_t nrhs,
                                const armpl_doublecomplex_t *ab,
                                armpl_int_t ldab, const armpl_int_t *ipiv,
                                armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgbtrs_work*, *LAPACKE_dgbtrs_work*, *LAPACKE_sgbtrs_work* and *LAPACKE_zgbtrs_work*.

4.19.1834 LAPACKE_zgebak

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgebak(armpl_int_t matrix_layout, char job, char side,
                           armpl_int_t n, armpl_int_t ilo, armpl_int_t ihi,
                           const double *scale, armpl_int_t m,
                           armpl_doublecomplex_t *v, armpl_int_t ldv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgebak*, *LAPACKE_dgebak*, *LAPACKE_sgebak* and *LAPACKE_zgebak*. It also exists with a native Fortran interface as *zgebak*.

4.19.1835 LAPACKE_zgebak_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgebak_work(armpl_int_t matrix_layout, char job,
                                char side, armpl_int_t n, armpl_int_t ilo,
                                armpl_int_t ihi, const double *scale,
                                armpl_int_t m, armpl_doublecomplex_t *v,
                                armpl_int_t ldv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgebak_work*, *LAPACKE_dgebak_work*, *LAPACKE_sgebak_work* and *LAPACKE_zgebak_work*.

4.19.1836 LAPACKE_zgebal

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgebal(armpl_int_t matrix_layout, char job, armpl_int_t n,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           armpl_int_t *ilo, armpl_int_t *ihi,
                           double *scale);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgebal*, *LAPACKE_dgebal*, *LAPACKE_sgebal* and *LAPACKE_zgebal*. It also exists with a native Fortran interface as *zgebal*.

4.19.1837 LAPACKE_zgebal_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgebal_work(armpl_int_t matrix_layout, char job,
                                armpl_int_t n, armpl_doublecomplex_t *a,
                                armpl_int_t lda, armpl_int_t *ilo,
                                armpl_int_t *ihi, double *scale);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgebal_work*, *LAPACKE_dgebal_work*, *LAPACKE_sgebal_work* and *LAPACKE_zgebal_work*.

4.19.1838 LAPACKE_zgebrd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgebrd(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_doublecomplex_t *a,
                           armpl_int_t lda, double *d, double *e,
                           armpl_doublecomplex_t *tauq,
                           armpl_doublecomplex_t *taup);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgebrd*, *LAPACKE_dgebrd*, *LAPACKE_sgebrd* and *LAPACKE_zgebrd*. It also exists with a native Fortran interface as *zgebrd*.

4.19.1839 LAPACKE_zgebrd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgebrd_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_doublecomplex_t *a,
                                armpl_int_t lda, double *d, double *e,
                                armpl_doublecomplex_t *tauq,
                                armpl_doublecomplex_t *taup,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgebrd_work*, *LAPACKE_dgebrd_work*, *LAPACKE_sgebrd_work* and *LAPACKE_zgebrd_work*.

4.19.1840 LAPACKE_zgecon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgecon(armpl_int_t matrix_layout, char norm,
                           armpl_int_t n, const armpl_doublecomplex_t *a,
                           armpl_int_t lda, double anorm, double *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgecon*, *LAPACKE_dgecon*, *LAPACKE_sgecon* and *LAPACKE_zgecon*. It also exists with a native Fortran interface as *zgecon*.

4.19.1841 LAPACKE_zgecon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgecon_work(armpl_int_t matrix_layout, char norm,
                                armpl_int_t n, const armpl_doublecomplex_t *a,
                                armpl_int_t lda, double anorm, double *rcond,
                                armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgecon_work*, *LAPACKE_dgecon_work*, *LAPACKE_sgecon_work* and *LAPACKE_zgecon_work*.

4.19.1842 LAPACKE_zgeequ

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgeequ(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, const armpl_doublecomplex_t *a,
                           armpl_int_t lda, double *r, double *c,
                           double *rowcnd, double *colcnd, double *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeequ*, *LAPACKE_dgeequ*, *LAPACKE_sgeequ* and *LAPACKE_zgeequ*. It also exists with a native Fortran interface as *zgeequ*.

4.19.1843 LAPACKE_zgeequ_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgeequ_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, const armpl_doublecomplex_t *a,
                                armpl_int_t lda, double *r, double *c,
                                double *rowcnd, double *colcnd,
                                double *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeequ_work*, *LAPACKE_dgeequ_work*, *LAPACKE_sgeequ_work* and *LAPACKE_zgeequ_work*.

4.19.1844 LAPACKE_zgeequb

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgeequb(armpl_int_t matrix_layout, armpl_int_t m,
                            armpl_int_t n, const armpl_doublecomplex_t *a,
                            armpl_int_t lda, double *r, double *c,
                            double *rowcnd, double *colcnd, double *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeequb*, *LAPACKE_dgeequb*, *LAPACKE_sgeequb* and *LAPACKE_zgeequb*. It also exists with a native Fortran interface as *zgeequb*.

4.19.1845 LAPACKE_zgeequb_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgeequb_work(armpl_int_t matrix_layout, armpl_int_t m,
                                 armpl_int_t n,
                                 const armpl_doublecomplex_t *a,
                                 armpl_int_t lda, double *r, double *c,
                                 double *rowcnd, double *colcnd,
                                 double *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeequb_work*, *LAPACKE_dgeequb_work*, *LAPACKE_sgeequb_work* and *LAPACKE_zgeequb_work*.

4.19.1846 LAPACKE_zgees

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgees(armpl_int_t matrix_layout, char jobvs, char sort,
                          LAPACK_Z_SELECT1 select, armpl_int_t n,
                          armpl_doublecomplex_t *a, armpl_int_t lda,
                          armpl_int_t *sdim, armpl_doublecomplex_t *w,
                          armpl_doublecomplex_t *vs, armpl_int_t ldvs);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgees*, *LAPACKE_dgees*, *LAPACKE_sgees* and *LAPACKE_zgees*. It also exists with a native Fortran interface as *zgees*.

4.19.1847 LAPACKE_zgees_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgees_work(armpl_int_t matrix_layout, char jobvs,
                               char sort, LAPACK_Z_SELECT1 select,
                               armpl_int_t n, armpl_doublecomplex_t *a,
                               armpl_int_t lda, armpl_int_t *sdim,
                               armpl_doublecomplex_t *w,
                               armpl_doublecomplex_t *vs, armpl_int_t ldvs,
                               armpl_doublecomplex_t *work, armpl_int_t lwork,
                               double *rwork, armpl_int_t *bwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgees_work*, *LAPACKE_dgees_work*, *LAPACKE_sgees_work* and *LAPACKE_zgees_work*.

4.19.1848 LAPACKE_zgeesx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgeesx(armpl_int_t matrix_layout, char jobvs, char sort,
                           LAPACK_Z_SELECT1 select, char sense, armpl_int_t n,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           armpl_int_t *sdim, armpl_doublecomplex_t *w,
                           armpl_doublecomplex_t *vs, armpl_int_t ldvs,
                           double *rconde, double *rcondv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeesx*, *LAPACKE_dgeesx*, *LAPACKE_sgeesx* and *LAPACKE_zgeesx*. It also exists with a native Fortran interface as *zgeesx*.

4.19.1849 LAPACKE_zgeesx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgeesx_work(armpl_int_t matrix_layout, char jobvs,
                                char sort, LAPACK_Z_SELECT1 select,
                                char sense, armpl_int_t n,
                                armpl_doublecomplex_t *a, armpl_int_t lda,
                                armpl_int_t *sdim, armpl_doublecomplex_t *w,
                                armpl_doublecomplex_t *vs, armpl_int_t ldvs,
                                double *rconde, double *rcondv,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork, double *rwork,
                                armpl_int_t *bwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeesx_work*, *LAPACKE_dgeesx_work*, *LAPACKE_sgeesx_work* and *LAPACKE_zgeesx_work*.

4.19.1850 LAPACKE_zgeev

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgeev(armpl_int_t matrix_layout, char jobvl, char jobvr,
                          armpl_int_t n, armpl_doublecomplex_t *a,
                          armpl_int_t lda, armpl_doublecomplex_t *w,
                          armpl_doublecomplex_t *vl, armpl_int_t ldvl,
                          armpl_doublecomplex_t *vr, armpl_int_t ldvr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeev*, *LAPACKE_dgeev*, *LAPACKE_sgeev* and *LAPACKE_zgeev*. It also exists with a native Fortran interface as *zgeev*.

4.19.1851 LAPACKE_zgeev_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgeev_work(armpl_int_t matrix_layout, char jobvl,
                               char jobvr, armpl_int_t n,
                               armpl_doublecomplex_t *a, armpl_int_t lda,
                               armpl_doublecomplex_t *w,
```

(continues on next page)

(continued from previous page)

```
                                armpl_doublecomplex_t *vl, armpl_int_t ldvl,
                                armpl_doublecomplex_t *vr, armpl_int_t ldvr,
                                armpl_doublecomplex_t *work, armpl_int_t lwork,
                                double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeev_work*, *LAPACKE_dgeev_work*, *LAPACKE_sgeev_work* and *LAPACKE_zgeev_work*.

4.19.1852 LAPACKE_zgeevx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgeevx(armpl_int_t matrix_layout, char balanc, char jobvl,
                           char jobvr, char sense, armpl_int_t n,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           armpl_doublecomplex_t *w,
                           armpl_doublecomplex_t *vl, armpl_int_t ldvl,
                           armpl_doublecomplex_t *vr, armpl_int_t ldvr,
                           armpl_int_t *ilo, armpl_int_t *ihi, double *scale,
                           double *abnrm, double *rconde, double *rcondv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeevx*, *LAPACKE_dgeevx*, *LAPACKE_sgeevx* and *LAPACKE_zgeevx*. It also exists with a native Fortran interface as *zgeevx*.

4.19.1853 LAPACKE_zgeevx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgeevx_work(armpl_int_t matrix_layout, char balanc,
                                char jobvl, char jobvr, char sense,
                                armpl_int_t n, armpl_doublecomplex_t *a,
                                armpl_int_t lda, armpl_doublecomplex_t *w,
                                armpl_doublecomplex_t *vl, armpl_int_t ldvl,
                                armpl_doublecomplex_t *vr, armpl_int_t ldvr,
                                armpl_int_t *ilo, armpl_int_t *ihi,
                                double *scale, double *abnrm, double *rconde,
                                double *rcondv, armpl_doublecomplex_t *work,
                                armpl_int_t lwork, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeevx_work*, *LAPACKE_dgeevx_work*, *LAPACKE_sgeevx_work* and *LAPACKE_zgeevx_work*.

4.19.1854 LAPACKE_zgehrd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgehrd(armpl_int_t matrix_layout, armpl_int_t n,
                           armpl_int_t ilo, armpl_int_t ihi,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           armpl_doublecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.1855 LAPACKE_zgehrd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgehrd_work(armpl_int_t matrix_layout, armpl_int_t n,
                                armpl_int_t ilo, armpl_int_t ihi,
                                armpl_doublecomplex_t *a, armpl_int_t lda,
                                armpl_doublecomplex_t *tau,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgehrd_work*, *LAPACKE_dgehrd_work*, *LAPACKE_sgehrd_work* and *LAPACKE_zgehrd_work*.

4.19.1856 LAPACKE_zgejsv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgejsv(armpl_int_t matrix_layout, char joba, char jobu,
                           char jobv, char jobr, char jobt, char jobp,
                           armpl_int_t m, armpl_int_t n,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           double *sva, armpl_doublecomplex_t *u,
                           armpl_int_t ldu, armpl_doublecomplex_t *v,
                           armpl_int_t ldv, double *stat,
                           armpl_int_t *istat);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgejsv*, *LAPACKE_dgejsv*, *LAPACKE_sgejsv* and *LAPACKE_zgejsv*. It also exists with a native Fortran interface as *zgejsv*.

4.19.1857 LAPACKE_zgejsv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgejsv_work(armpl_int_t matrix_layout, char joba,
                                char jobu, char jobv, char jobr, char jobt,
                                char jobp, armpl_int_t m, armpl_int_t n,
                                armpl_doublecomplex_t *a, armpl_int_t lda,
                                double *sva, armpl_doublecomplex_t *u,
                                armpl_int_t ldu, armpl_doublecomplex_t *v,
                                armpl_int_t ldv, armpl_doublecomplex_t *cwork,
                                armpl_int_t lwork, double *work,
                                armpl_int_t lrwork, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgejsv_work*, *LAPACKE_dgejsv_work*, *LAPACKE_sgejsv_work* and *LAPACKE_zgejsv_work*.

4.19.1858 LAPACKE_zgelq

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgelq(armpl_int_t matrix_layout, armpl_int_t m,
                          armpl_int_t n, armpl_doublecomplex_t *a,
                          armpl_int_t lda, armpl_doublecomplex_t *t,
                          armpl_int_t tsize);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelq*, *LAPACKE_dgelq*, *LAPACKE_sgelq* and *LAPACKE_zgelq*. It also exists with a native Fortran interface as *zgelq*.

4.19.1859 LAPACKE_zgelq2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgelq2(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_doublecomplex_t *a,
                           armpl_int_t lda, armpl_doublecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelq2*, *LAPACKE_dgelq2*, *LAPACKE_sgelq2* and *LAPACKE_zgelq2*. It also exists with a native Fortran interface as *zgelq2*.

4.19.1860 LAPACKE_zgelq2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgelq2_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_doublecomplex_t *a,
                                armpl_int_t lda, armpl_doublecomplex_t *tau,
                                armpl_doublecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelq2_work*, *LAPACKE_dgelq2_work*, *LAPACKE_sgelq2_work* and *LAPACKE_zgelq2_work*.

4.19.1861 LAPACKE_zgelq_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgelq_work(armpl_int_t matrix_layout, armpl_int_t m,
                               armpl_int_t n, armpl_doublecomplex_t *a,
                               armpl_int_t lda, armpl_doublecomplex_t *t,
                               armpl_int_t tsize, armpl_doublecomplex_t *work,
                               armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelq_work*, *LAPACKE_dgelq_work*, *LAPACKE_sgelq_work* and *LAPACKE_zgelq_work*.

4.19.1862 LAPACKE_zgelqf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgelqf(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_doublecomplex_t *a,
                           armpl_int_t lda, armpl_doublecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelqf*, *LAPACKE_dgelqf*, *LAPACKE_sgelqf* and *LAPACKE_zgelqf*. It also exists with a native Fortran interface as *zgelqf*.

4.19.1863 LAPACKE_zgelqf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgelqf_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_doublecomplex_t *a,
                                armpl_int_t lda, armpl_doublecomplex_t *tau,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelqf_work*, *LAPACKE_dgelqf_work*, *LAPACKE_sgelqf_work* and *LAPACKE_zgelqf_work*.

4.19.1864 LAPACKE_zgels

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgels(armpl_int_t matrix_layout, char trans,
                          armpl_int_t m, armpl_int_t n, armpl_int_t nrhs,
                          armpl_doublecomplex_t *a, armpl_int_t lda,
                          armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgels*, *LAPACKE_dgels*, *LAPACKE_sgels* and *LAPACKE_zgels*. It also exists with a native Fortran interface as *zgels*.

4.19.1865 LAPACKE_zgels_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgels_work(armpl_int_t matrix_layout, char trans,
                               armpl_int_t m, armpl_int_t n, armpl_int_t nrhs,
                               armpl_doublecomplex_t *a, armpl_int_t lda,
                               armpl_doublecomplex_t *b, armpl_int_t ldb,
                               armpl_doublecomplex_t *work,
                               armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgels_work*, *LAPACKE_dgels_work*, *LAPACKE_sgels_work* and *LAPACKE_zgels_work*.

4.19.1866 LAPACKE_zgelsd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgelsd(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t nrhs,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           armpl_doublecomplex_t *b, armpl_int_t ldb,
                           double *s, double rcond, armpl_int_t *rank);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelsd*, *LAPACKE_dgelsd*, *LAPACKE_sgelsd* and *LAPACKE_zgelsd*. It also exists with a native Fortran interface as *zgelsd*.

4.19.1867 LAPACKE_zgelsd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgelsd_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t nrhs,
                                armpl_doublecomplex_t *a, armpl_int_t lda,
                                armpl_doublecomplex_t *b, armpl_int_t ldb,
                                double *s, double rcond, armpl_int_t *rank,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork, double *rwork,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelsd_work*, *LAPACKE_dgelsd_work*, *LAPACKE_sgelsd_work* and *LAPACKE_zgelsd_work*.

4.19.1868 LAPACKE_zgelss

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgelss(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t nrhs,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           armpl_doublecomplex_t *b, armpl_int_t ldb,
                           double *s, double rcond, armpl_int_t *rank);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelss*, *LAPACKE_dgelss*, *LAPACKE_sgelss* and *LAPACKE_zgelss*. It also exists with a native Fortran interface as *zgelss*.

4.19.1869 LAPACKE_zgelss_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgelss_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t nrhs,
                                armpl_doublecomplex_t *a, armpl_int_t lda,
                                armpl_doublecomplex_t *b, armpl_int_t ldb,
                                double *s, double rcond, armpl_int_t *rank,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelss_work*, *LAPACKE_dgelss_work*, *LAPACKE_sgelss_work* and *LAPACKE_zgelss_work*.

4.19.1870 LAPACKE_zgelsy

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgelsy(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t nrhs,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           armpl_doublecomplex_t *b, armpl_int_t ldb,
                           armpl_int_t *jpvt, double rcond,
                           armpl_int_t *rank);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelsy*, *LAPACKE_dgelsy*, *LAPACKE_sgelsy* and *LAPACKE_zgelsy*. It also exists with a native Fortran interface as *zgelsy*.

4.19.1871 LAPACKE_zgelsy_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgelsy_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t nrhs,
                                armpl_doublecomplex_t *a, armpl_int_t lda,
                                armpl_doublecomplex_t *b, armpl_int_t ldb,
                                armpl_int_t *jpvt, double rcond,
                                armpl_int_t *rank,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgelsy_work*, *LAPACKE_dgelsy_work*, *LAPACKE_sgelsy_work* and *LAPACKE_zgelsy_work*.

4.19.1872 LAPACKE_zgemlq

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgemlq(armpl_int_t matrix_layout, char side, char trans,
                           armpl_int_t m, armpl_int_t n, armpl_int_t k,
                           const armpl_doublecomplex_t *a, armpl_int_t lda,
                           const armpl_doublecomplex_t *t, armpl_int_t tsize,
                           armpl_doublecomplex_t *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgemlq*, *LAPACKE_dgemlq*, *LAPACKE_sgemlq* and *LAPACKE_zgemlq*. It also exists with a native Fortran interface as *zgemlq*.

4.19.1873 LAPACKE_zgemlq_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgemlq_work(armpl_int_t matrix_layout, char side,
                                char trans, armpl_int_t m, armpl_int_t n,
                                armpl_int_t k, const armpl_doublecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_doublecomplex_t *t,
                                armpl_int_t tsize, armpl_doublecomplex_t *c,
                                armpl_int_t ldc, armpl_doublecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgemlq_work*, *LAPACKE_dgemlq_work*, *LAPACKE_sgemlq_work* and *LAPACKE_zgemlq_work*.

4.19.1874 LAPACKE_zgemqr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgemqr(armpl_int_t matrix_layout, char side, char trans,
                           armpl_int_t m, armpl_int_t n, armpl_int_t k,
                           const armpl_doublecomplex_t *a, armpl_int_t lda,
                           const armpl_doublecomplex_t *t, armpl_int_t tsize,
                           armpl_doublecomplex_t *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgemqr*, *LAPACKE_dgemqr*, *LAPACKE_sgemqr* and *LAPACKE_zgemqr*. It also exists with a native Fortran interface as *zgemqr*.

4.19.1875 LAPACKE_zgemqr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgemqr_work(armpl_int_t matrix_layout, char side,
                                char trans, armpl_int_t m, armpl_int_t n,
                                armpl_int_t k, const armpl_doublecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_doublecomplex_t *t,
                                armpl_int_t tsize, armpl_doublecomplex_t *c,
                                armpl_int_t ldc, armpl_doublecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgemqr_work*, *LAPACKE_dgemqr_work*, *LAPACKE_sgemqr_work* and *LAPACKE_zgemqr_work*.

4.19.1876 LAPACKE_zgemqrt

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgemqrt(armpl_int_t matrix_layout, char side, char trans,
                            armpl_int_t m, armpl_int_t n, armpl_int_t k,
                            armpl_int_t nb, const armpl_doublecomplex_t *v,
                            armpl_int_t ldv, const armpl_doublecomplex_t *t,
                            armpl_int_t ldt, armpl_doublecomplex_t *c,
                            armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgemqrt*, *LAPACKE_dgemqrt*, *LAPACKE_sgemqrt* and *LAPACKE_zgemqrt*. It also exists with a native Fortran interface as *zgemqrt*.

4.19.1877 LAPACKE_zgemqrt_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgemqrt_work(armpl_int_t matrix_layout, char side,
                                 char trans, armpl_int_t m, armpl_int_t n,
                                 armpl_int_t k, armpl_int_t nb,
```

(continues on next page)

(continued from previous page)

```
                                const armpl_doublecomplex_t *v,
                                armpl_int_t ldv,
                                const armpl_doublecomplex_t *t,
                                armpl_int_t ldt, armpl_doublecomplex_t *c,
                                armpl_int_t ldc,
                                armpl_doublecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgemqrt_work*, *LAPACKE_dgemqrt_work*, *LAPACKE_sgemqrt_work* and *LAPACKE_zgemqrt_work*.

4.19.1878 LAPACKE_zgeqlf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgeqlf(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_doublecomplex_t *a,
                           armpl_int_t lda, armpl_doublecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqlf*, *LAPACKE_dgeqlf*, *LAPACKE_sgeqlf* and *LAPACKE_zgeqlf*. It also exists with a native Fortran interface as *zgeqlf*.

4.19.1879 LAPACKE_zgeqlf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgeqlf_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_doublecomplex_t *a,
                                armpl_int_t lda, armpl_doublecomplex_t *tau,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqlf_work*, *LAPACKE_dgeqlf_work*, *LAPACKE_sgeqlf_work* and *LAPACKE_zgeqlf_work*.

4.19.1880 LAPACKE_zgeqp3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgeqp3(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_doublecomplex_t *a,
                           armpl_int_t lda, armpl_int_t *jpvt,
                           armpl_doublecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqp3*, *LAPACKE_dgeqp3*, *LAPACKE_sgeqp3* and *LAPACKE_zgeqp3*. It also exists with a native Fortran interface as *zgeqp3*.

4.19.1881 LAPACKE_zgeqp3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgeqp3_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_doublecomplex_t *a,
                                armpl_int_t lda, armpl_int_t *jpvt,
                                armpl_doublecomplex_t *tau,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqp3_work*, *LAPACKE_dgeqp3_work*, *LAPACKE_sgeqp3_work* and *LAPACKE_zgeqp3_work*.

4.19.1882 LAPACKE_zgeqpf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgeqpf(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_doublecomplex_t *a,
                           armpl_int_t lda, armpl_int_t *jpvt,
                           armpl_doublecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqpf*, *LAPACKE_dgeqpf*, *LAPACKE_sgeqpf* and *LAPACKE_zgeqpf*.

4.19.1883 LAPACKE_zgeqpf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgeqpf_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_doublecomplex_t *a,
                                armpl_int_t lda, armpl_int_t *jpvt,
                                armpl_doublecomplex_t *tau,
                                armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqpf_work*, *LAPACKE_dgeqpf_work*, *LAPACKE_sgeqpf_work* and *LAPACKE_zgeqpf_work*.

4.19.1884 LAPACKE_zgeqr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgeqr(armpl_int_t matrix_layout, armpl_int_t m,
                          armpl_int_t n, armpl_doublecomplex_t *a,
                          armpl_int_t lda, armpl_doublecomplex_t *t,
                          armpl_int_t tsize);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqr*, *LAPACKE_dgeqr*, *LAPACKE_sgeqr* and *LAPACKE_zgeqr*. It also exists with a native Fortran interface as *zgeqr*.

4.19.1885 LAPACKE_zgeqr2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgeqr2(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_doublecomplex_t *a,
                           armpl_int_t lda, armpl_doublecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqr2*, *LAPACKE_dgeqr2*, *LAPACKE_sgeqr2* and *LAPACKE_zgeqr2*. It also exists with a native Fortran interface as *zgeqr2*.

4.19.1886 LAPACKE_zgeqr2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgeqr2_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_doublecomplex_t *a,
                                armpl_int_t lda, armpl_doublecomplex_t *tau,
                                armpl_doublecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqr2_work*, *LAPACKE_dgeqr2_work*, *LAPACKE_sgeqr2_work* and *LAPACKE_zgeqr2_work*.

4.19.1887 LAPACKE_zgeqr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgeqr_work(armpl_int_t matrix_layout, armpl_int_t m,
                               armpl_int_t n, armpl_doublecomplex_t *a,
                               armpl_int_t lda, armpl_doublecomplex_t *t,
                               armpl_int_t tsize, armpl_doublecomplex_t *work,
                               armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqr_work*, *LAPACKE_dgeqr_work*, *LAPACKE_sgeqr_work* and *LAPACKE_zgeqr_work*.

4.19.1888 LAPACKE_zgeqrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgeqrf(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_doublecomplex_t *a,
                           armpl_int_t lda, armpl_doublecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrf*, *LAPACKE_dgeqrf*, *LAPACKE_sgeqrf* and *LAPACKE_zgeqrf*. It also exists with a native Fortran interface as *zgeqrf*.

4.19.1889 LAPACKE_zgeqrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgeqrf_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_doublecomplex_t *a,
                                armpl_int_t lda, armpl_doublecomplex_t *tau,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrf_work*, *LAPACKE_dgeqrf_work*, *LAPACKE_sgeqrf_work* and *LAPACKE_zgeqrf_work*.

4.19.1890 LAPACKE_zgeqrfp

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgeqrfp(armpl_int_t matrix_layout, armpl_int_t m,
                            armpl_int_t n, armpl_doublecomplex_t *a,
                            armpl_int_t lda, armpl_doublecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrfp*, *LAPACKE_dgeqrfp*, *LAPACKE_sgeqrfp* and *LAPACKE_zgeqrfp*. It also exists with a native Fortran interface as *zgeqrfp*.

4.19.1891 LAPACKE_zgeqrfp_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgeqrfp_work(armpl_int_t matrix_layout, armpl_int_t m,
                                 armpl_int_t n, armpl_doublecomplex_t *a,
                                 armpl_int_t lda, armpl_doublecomplex_t *tau,
                                 armpl_doublecomplex_t *work,
                                 armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrfp_work*, *LAPACKE_dgeqrfp_work*, *LAPACKE_sgeqrfp_work* and *LAPACKE_zgeqrfp_work*.

4.19.1892 LAPACKE_zgeqrt

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgeqrt(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t nb,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           armpl_doublecomplex_t *t, armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrt*, *LAPACKE_dgeqrt*, *LAPACKE_sgeqrt* and *LAPACKE_zgeqrt*. It also exists with a native Fortran interface as *zgeqrt*.

4.19.1893 LAPACKE_zgeqrt2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgeqrt2(armpl_int_t matrix_layout, armpl_int_t m,
                            armpl_int_t n, armpl_doublecomplex_t *a,
                            armpl_int_t lda, armpl_doublecomplex_t *t,
                            armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrt2*, *LAPACKE_dgeqrt2*, *LAPACKE_sgeqrt2* and *LAPACKE_zgeqrt2*. It also exists with a native Fortran interface as *zgeqrt2*.

4.19.1894 LAPACKE_zgeqrt2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgeqrt2_work(armpl_int_t matrix_layout, armpl_int_t m,
                                 armpl_int_t n, armpl_doublecomplex_t *a,
                                 armpl_int_t lda, armpl_doublecomplex_t *t,
                                 armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrt2_work*, *LAPACKE_dgeqrt2_work*, *LAPACKE_sgeqrt2_work* and *LAPACKE_zgeqrt2_work*.

4.19.1895 LAPACKE_zgeqrt3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgeqrt3(armpl_int_t matrix_layout, armpl_int_t m,
                            armpl_int_t n, armpl_doublecomplex_t *a,
                            armpl_int_t lda, armpl_doublecomplex_t *t,
                            armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrt3*, *LAPACKE_dgeqrt3*, *LAPACKE_sgeqrt3* and *LAPACKE_zgeqrt3*. It also exists with a native Fortran interface as *zgeqrt3*.

4.19.1896 LAPACKE_zgeqrt3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgeqrt3_work(armpl_int_t matrix_layout, armpl_int_t m,
                                 armpl_int_t n, armpl_doublecomplex_t *a,
                                 armpl_int_t lda, armpl_doublecomplex_t *t,
                                 armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrt3_work*, *LAPACKE_dgeqrt3_work*, *LAPACKE_sgeqrt3_work* and *LAPACKE_zgeqrt3_work*.

4.19.1897 LAPACKE_zgeqrt_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgeqrt_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t nb,
                                armpl_doublecomplex_t *a, armpl_int_t lda,
                                armpl_doublecomplex_t *t, armpl_int_t ldt,
                                armpl_doublecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgeqrt_work*, *LAPACKE_dgeqrt_work*, *LAPACKE_sgeqrt_work* and *LAPACKE_zgeqrt_work*.

4.19.1898 LAPACKE_zgerfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgerfs(armpl_int_t matrix_layout, char trans,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_doublecomplex_t *a, armpl_int_t lda,
                           const armpl_doublecomplex_t *af, armpl_int_t ldaf,
                           const armpl_int_t *ipiv,
                           const armpl_doublecomplex_t *b, armpl_int_t ldb,
                           armpl_doublecomplex_t *x, armpl_int_t ldx,
                           double *ferr, double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgerfs*, *LAPACKE_dgerfs*, *LAPACKE_sgerfs* and *LAPACKE_zgerfs*. It also exists with a native Fortran interface as *zgerfs*.

4.19.1899 LAPACKE_zgerfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgerfs_work(armpl_int_t matrix_layout, char trans,
                                armpl_int_t n, armpl_int_t nrhs,
                                const armpl_doublecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_doublecomplex_t *af,
                                armpl_int_t ldaf, const armpl_int_t *ipiv,
                                const armpl_doublecomplex_t *b,
                                armpl_int_t ldb, armpl_doublecomplex_t *x,
                                armpl_int_t ldx, double *ferr, double *berr,
                                armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgerfs_work*, *LAPACKE_dgerfs_work*, *LAPACKE_sgerfs_work* and *LAPACKE_zgerfs_work*.

4.19.1900 LAPACKE_zgerfsx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgerfsx(armpl_int_t matrix_layout, char trans, char equed,
                            armpl_int_t n, armpl_int_t nrhs,
                            const armpl_doublecomplex_t *a, armpl_int_t lda,
                            const armpl_doublecomplex_t *af, armpl_int_t ldaf,
                            const armpl_int_t *ipiv, const double *r,
                            const double *c, const armpl_doublecomplex_t *b,
                            armpl_int_t ldb, armpl_doublecomplex_t *x,
                            armpl_int_t ldx, double *rcond, double *berr,
                            armpl_int_t n_err_bnds, double *err_bnds_norm,
                            double *err_bnds_comp, armpl_int_t nparams,
                            double *params);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgerfsx*, *LAPACKE_dgerfsx*, *LAPACKE_sgerfsx* and *LAPACKE_zgerfsx*. It also exists with a native Fortran interface as *zgerfsx*.

4.19.1901 LAPACKE_zgerfsx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgerfsx_work(armpl_int_t matrix_layout, char trans,
                                 char equed, armpl_int_t n, armpl_int_t nrhs,
                                 const armpl_doublecomplex_t *a,
                                 armpl_int_t lda,
                                 const armpl_doublecomplex_t *af,
                                 armpl_int_t ldaf, const armpl_int_t *ipiv,
                                 const double *r, const double *c,
                                 const armpl_doublecomplex_t *b,
                                 armpl_int_t ldb, armpl_doublecomplex_t *x,
                                 armpl_int_t ldx, double *rcond, double *berr,
                                 armpl_int_t n_err_bnds,
                                 double *err_bnds_norm, double *err_bnds_comp,
                                 armpl_int_t nparams, double *params,
                                 armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgerfsx_work*, *LAPACKE_dgerfsx_work*, *LAPACKE_sgerfsx_work* and *LAPACKE_zgerfsx_work*.

4.19.1902 LAPACKE_zgerqf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgerqf(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_doublecomplex_t *a,
                           armpl_int_t lda, armpl_doublecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgerqf*, *LAPACKE_dgerqf*, *LAPACKE_sgerqf* and *LAPACKE_zgerqf*. It also exists with a native Fortran interface as *zgerqf*.

4.19.1903 LAPACKE_zgerqf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgerqf_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_doublecomplex_t *a,
                                armpl_int_t lda, armpl_doublecomplex_t *tau,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgerqf_work*, *LAPACKE_dgerqf_work*, *LAPACKE_sgerqf_work* and *LAPACKE_zgerqf_work*.

4.19.1904 LAPACKE_zgesdd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgesdd(armpl_int_t matrix_layout, char jobz,
                           armpl_int_t m, armpl_int_t n,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           double *s, armpl_doublecomplex_t *u,
                           armpl_int_t ldu, armpl_doublecomplex_t *vt,
                           armpl_int_t ldvt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesdd*, *LAPACKE_dgesdd*, *LAPACKE_sgesdd* and *LAPACKE_zgesdd*. It also exists with a native Fortran interface as *zgesdd*.

4.19.1905 LAPACKE_zgesdd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgesdd_work(armpl_int_t matrix_layout, char jobz,
                                armpl_int_t m, armpl_int_t n,
                                armpl_doublecomplex_t *a, armpl_int_t lda,
                                double *s, armpl_doublecomplex_t *u,
                                armpl_int_t ldu, armpl_doublecomplex_t *vt,
                                armpl_int_t ldvt, armpl_doublecomplex_t *work,
                                armpl_int_t lwork, double *rwork,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesdd_work*, *LAPACKE_dgesdd_work*, *LAPACKE_sgesdd_work* and *LAPACKE_zgesdd_work*.

4.19.1906 LAPACKE_zgesv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgesv(armpl_int_t matrix_layout, armpl_int_t n,
                          armpl_int_t nrhs, armpl_doublecomplex_t *a,
                          armpl_int_t lda, armpl_int_t *ipiv,
                          armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesv*, *LAPACKE_dgesv*, *LAPACKE_sgesv* and *LAPACKE_zgesv*. It also exists with a native Fortran interface as *zgesv*.

4.19.1907 LAPACKE_zgesv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgesv_work(armpl_int_t matrix_layout, armpl_int_t n,
                               armpl_int_t nrhs, armpl_doublecomplex_t *a,
                               armpl_int_t lda, armpl_int_t *ipiv,
                               armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesv_work*, *LAPACKE_dgesv_work*, *LAPACKE_sgesv_work* and *LAPACKE_zgesv_work*.

4.19.1908 LAPACKE_zgesvd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgesvd(armpl_int_t matrix_layout, char jobu, char jobvt,
                           armpl_int_t m, armpl_int_t n,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           double *s, armpl_doublecomplex_t *u,
                           armpl_int_t ldu, armpl_doublecomplex_t *vt,
                           armpl_int_t ldvt, double *superb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvd*, *LAPACKE_dgesvd*, *LAPACKE_sgesvd* and *LAPACKE_zgesvd*. It also exists with a native Fortran interface as *zgesvd*.

4.19.1909 LAPACKE_zgesvd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgesvd_work(armpl_int_t matrix_layout, char jobu,
                                char jobvt, armpl_int_t m, armpl_int_t n,
                                armpl_doublecomplex_t *a, armpl_int_t lda,
                                double *s, armpl_doublecomplex_t *u,
                                armpl_int_t ldu, armpl_doublecomplex_t *vt,
                                armpl_int_t ldvt, armpl_doublecomplex_t *work,
                                armpl_int_t lwork, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvd_work*, *LAPACKE_dgesvd_work*, *LAPACKE_sgesvd_work* and *LAPACKE_zgesvd_work*.

4.19.1910 LAPACKE_zgesvdx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgesvdx(armpl_int_t matrix_layout, char jobu, char jobvt,
                            char range, armpl_int_t m, armpl_int_t n,
                            armpl_doublecomplex_t *a, armpl_int_t lda,
                            double vl, double vu, armpl_int_t il,
                            armpl_int_t iu, armpl_int_t *ns, double *s,
                            armpl_doublecomplex_t *u, armpl_int_t ldu,
                            armpl_doublecomplex_t *vt, armpl_int_t ldvt,
                            armpl_int_t *superb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvdx*, *LAPACKE_dgesvdx*, *LAPACKE_sgesvdx* and *LAPACKE_zgesvdx*. It also exists with a native Fortran interface as *zgesvdx*.

4.19.1911 LAPACKE_zgesvdx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgesvdx_work(armpl_int_t matrix_layout, char jobu,
                                 char jobvt, char range, armpl_int_t m,
                                 armpl_int_t n, armpl_doublecomplex_t *a,
                                 armpl_int_t lda, double vl, double vu,
                                 armpl_int_t il, armpl_int_t iu,
                                 armpl_int_t *ns, double *s,
                                 armpl_doublecomplex_t *u, armpl_int_t ldu,
                                 armpl_doublecomplex_t *vt, armpl_int_t ldvt,
```

(continues on next page)

(continued from previous page)

```
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork, double *rwork,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvdx_work*, *LAPACKE_dgesvdx_work*, *LAPACKE_sgesvdx_work* and *LAPACKE_zgesvdx_work*.

4.19.1912 LAPACKE_zgesvj

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgesvj(armpl_int_t matrix_layout, char joba, char jobu,
                           char jobv, armpl_int_t m, armpl_int_t n,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           double *sva, armpl_int_t mv,
                           armpl_doublecomplex_t *v, armpl_int_t ldv,
                           double *stat);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvj*, *LAPACKE_dgesvj*, *LAPACKE_sgesvj* and *LAPACKE_zgesvj*. It also exists with a native Fortran interface as *zgesvj*.

4.19.1913 LAPACKE_zgesvj_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgesvj_work(armpl_int_t matrix_layout, char joba,
                                char jobu, char jobv, armpl_int_t m,
                                armpl_int_t n, armpl_doublecomplex_t *a,
                                armpl_int_t lda, double *sva, armpl_int_t mv,
                                armpl_doublecomplex_t *v, armpl_int_t ldv,
                                armpl_doublecomplex_t *cwork,
                                armpl_int_t lwork, double *rwork,
                                armpl_int_t lrwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvj_work*, *LAPACKE_dgesvj_work*, *LAPACKE_sgesvj_work* and *LAPACKE_zgesvj_work*.

4.19.1914 LAPACKE_zgesvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgesvx(armpl_int_t matrix_layout, char fact, char trans,
                           armpl_int_t n, armpl_int_t nrhs,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           armpl_doublecomplex_t *af, armpl_int_t ldaf,
                           armpl_int_t *ipiv, char *equed, double *r,
                           double *c, armpl_doublecomplex_t *b,
                           armpl_int_t ldb, armpl_doublecomplex_t *x,
                           armpl_int_t ldx, double *rcond, double *ferr,
                           double *berr, double *rpivot);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvx*, *LAPACKE_dgesvx*, *LAPACKE_sgesvx* and *LAPACKE_zgesvx*. It also exists with a native Fortran interface as *zgesvx*.

4.19.1915 LAPACKE_zgesvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgesvx_work(armpl_int_t matrix_layout, char fact,
                                char trans, armpl_int_t n, armpl_int_t nrhs,
                                armpl_doublecomplex_t *a, armpl_int_t lda,
                                armpl_doublecomplex_t *af, armpl_int_t ldaf,
                                armpl_int_t *ipiv, char *equed, double *r,
                                double *c, armpl_doublecomplex_t *b,
                                armpl_int_t ldb, armpl_doublecomplex_t *x,
                                armpl_int_t ldx, double *rcond, double *ferr,
                                double *berr, armpl_doublecomplex_t *work,
                                double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvx_work*, *LAPACKE_dgesvx_work*, *LAPACKE_sgesvx_work* and *LAPACKE_zgesvx_work*.

4.19.1916 LAPACKE_zgesvxx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgesvxx(armpl_int_t matrix_layout, char fact, char trans,
                            armpl_int_t n, armpl_int_t nrhs,
                            armpl_doublecomplex_t *a, armpl_int_t lda,
                            armpl_doublecomplex_t *af, armpl_int_t ldaf,
                            armpl_int_t *ipiv, char *equed, double *r,
                            double *c, armpl_doublecomplex_t *b,
```

(continues on next page)

(continued from previous page)

```
                                armpl_int_t ldb, armpl_doublecomplex_t *x,
                                armpl_int_t ldx, double *rcond, double *rpvgrw,
                                double *berr, armpl_int_t n_err_bnds,
                                double *err_bnds_norm, double *err_bnds_comp,
                                armpl_int_t nparams, double *params);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvxx*, *LAPACKE_dgesvxx*, *LAPACKE_sgesvxx* and *LAPACKE_zgesvxx*. It also exists with a native Fortran interface as *zgesvxx*.

4.19.1917 LAPACKE_zgesvxx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgesvxx_work(armpl_int_t matrix_layout, char fact,
                                 char trans, armpl_int_t n, armpl_int_t nrhs,
                                 armpl_doublecomplex_t *a, armpl_int_t lda,
                                 armpl_doublecomplex_t *af, armpl_int_t ldaf,
                                 armpl_int_t *ipiv, char *equed, double *r,
                                 double *c, armpl_doublecomplex_t *b,
                                 armpl_int_t ldb, armpl_doublecomplex_t *x,
                                 armpl_int_t ldx, double *rcond,
                                 double *rpvgrw, double *berr,
                                 armpl_int_t n_err_bnds,
                                 double *err_bnds_norm, double *err_bnds_comp,
                                 armpl_int_t nparams, double *params,
                                 armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgesvxx_work*, *LAPACKE_dgesvxx_work*, *LAPACKE_sgesvxx_work* and *LAPACKE_zgesvxx_work*.

4.19.1918 LAPACKE_zgetf2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgetf2(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_doublecomplex_t *a,
                           armpl_int_t lda, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetf2*, *LAPACKE_dgetf2*, *LAPACKE_sgetf2* and *LAPACKE_zgetf2*. It also exists with a native Fortran interface as *zgetf2*.

4.19.1919 LAPACKE_zgetf2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgetf2_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_doublecomplex_t *a,
                                armpl_int_t lda, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetf2_work*, *LAPACKE_dgetf2_work*, *LAPACKE_sgetf2_work* and *LAPACKE_zgetf2_work*.

4.19.1920 LAPACKE_zgetrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgetrf(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_doublecomplex_t *a,
                           armpl_int_t lda, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetrf*, *LAPACKE_dgetrf*, *LAPACKE_sgetrf* and *LAPACKE_zgetrf*. It also exists with a native Fortran interface as *zgetrf*.

4.19.1921 LAPACKE_zgetrf2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgetrf2(armpl_int_t matrix_layout, armpl_int_t m,
                            armpl_int_t n, armpl_doublecomplex_t *a,
                            armpl_int_t lda, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetrf2*, *LAPACKE_dgetrf2*, *LAPACKE_sgetrf2* and *LAPACKE_zgetrf2*. It also exists with a native Fortran interface as *zgetrf2*.

4.19.1922 LAPACKE_zgetrf2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgetrf2_work(armpl_int_t matrix_layout, armpl_int_t m,
                                 armpl_int_t n, armpl_doublecomplex_t *a,
                                 armpl_int_t lda, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetrf2_work*, *LAPACKE_dgetrf2_work*, *LAPACKE_sgetrf2_work* and *LAPACKE_zgetrf2_work*.

4.19.1923 LAPACKE_zgetrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgetrf_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_doublecomplex_t *a,
                                armpl_int_t lda, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetrf_work*, *LAPACKE_dgetrf_work*, *LAPACKE_sgetrf_work* and *LAPACKE_zgetrf_work*.

4.19.1924 LAPACKE_zgetri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgetri(armpl_int_t matrix_layout, armpl_int_t n,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           const armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetri*, *LAPACKE_dgetri*, *LAPACKE_sgetri* and *LAPACKE_zgetri*. It also exists with a native Fortran interface as *zgetri*.

4.19.1925 LAPACKE_zgetri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgetri_work(armpl_int_t matrix_layout, armpl_int_t n,
                                armpl_doublecomplex_t *a, armpl_int_t lda,
                                const armpl_int_t *ipiv,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetri_work*, *LAPACKE_dgetri_work*, *LAPACKE_sgetri_work* and *LAPACKE_zgetri_work*.

4.19.1926 LAPACKE_zgetrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgetrs(armpl_int_t matrix_layout, char trans,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_doublecomplex_t *a, armpl_int_t lda,
                           const armpl_int_t *ipiv, armpl_doublecomplex_t *b,
                           armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetrs*, *LAPACKE_dgetrs*, *LAPACKE_sgetrs* and *LAPACKE_zgetrs*. It also exists with a native Fortran interface as *zgetrs*.

4.19.1927 LAPACKE_zgetrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgetrs_work(armpl_int_t matrix_layout, char trans,
                                armpl_int_t n, armpl_int_t nrhs,
                                const armpl_doublecomplex_t *a,
                                armpl_int_t lda, const armpl_int_t *ipiv,
                                armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetrs_work*, *LAPACKE_dgetrs_work*, *LAPACKE_sgetrs_work* and *LAPACKE_zgetrs_work*.

4.19.1928 LAPACKE_zgetsls

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgetsls(armpl_int_t matrix_layout, char trans,
                            armpl_int_t m, armpl_int_t n, armpl_int_t nrhs,
                            armpl_doublecomplex_t *a, armpl_int_t lda,
                            armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetsls*, *LAPACKE_dgetsls*, *LAPACKE_sgetsls* and *LAPACKE_zgetsls*. It also exists with a native Fortran interface as *zgetsls*.

4.19.1929 LAPACKE_zgetsls_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgetsls_work(armpl_int_t matrix_layout, char trans,
                                 armpl_int_t m, armpl_int_t n,
                                 armpl_int_t nrhs, armpl_doublecomplex_t *a,
                                 armpl_int_t lda, armpl_doublecomplex_t *b,
                                 armpl_int_t ldb, armpl_doublecomplex_t *work,
                                 armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgetsls_work*, *LAPACKE_dgetsls_work*, *LAPACKE_sgetsls_work* and *LAPACKE_zgetsls_work*.

4.19.1930 LAPACKE_zggbak

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zggbak(armpl_int_t matrix_layout, char job, char side,
                           armpl_int_t n, armpl_int_t ilo, armpl_int_t ihi,
                           const double *lscale, const double *rscale,
                           armpl_int_t m, armpl_doublecomplex_t *v,
                           armpl_int_t ldv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggbak*, *LAPACKE_dggbak*, *LAPACKE_sggbak* and *LAPACKE_zggbak*. It also exists with a native Fortran interface as *zggbak*.

4.19.1931 LAPACKE_zggbak_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zggbak_work(armpl_int_t matrix_layout, char job,
                                char side, armpl_int_t n, armpl_int_t ilo,
                                armpl_int_t ihi, const double *lscale,
                                const double *rscale, armpl_int_t m,
                                armpl_doublecomplex_t *v, armpl_int_t ldv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggbak_work*, *LAPACKE_dggbak_work*, *LAPACKE_sggbak_work* and *LAPACKE_zggbak_work*.

4.19.1932 LAPACKE_zggbal

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zggbal(armpl_int_t matrix_layout, char job, armpl_int_t n,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           armpl_doublecomplex_t *b, armpl_int_t ldb,
                           armpl_int_t *ilo, armpl_int_t *ihi, double *lscale,
                           double *rscale);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggbal*, *LAPACKE_dggbal*, *LAPACKE_sggbal* and *LAPACKE_zggbal*. It also exists with a native Fortran interface as *zggbal*.

4.19.1933 LAPACKE_zggbal_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zggbal_work(armpl_int_t matrix_layout, char job,
                                armpl_int_t n, armpl_doublecomplex_t *a,
                                armpl_int_t lda, armpl_doublecomplex_t *b,
                                armpl_int_t ldb, armpl_int_t *ilo,
                                armpl_int_t *ihi, double *lscale,
                                double *rscale, double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggbal_work*, *LAPACKE_dggbal_work*, *LAPACKE_sggbal_work* and *LAPACKE_zggbal_work*.

4.19.1934 LAPACKE_zgges

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgges(armpl_int_t matrix_layout, char jobvsl, char jobvsr,
                          char sort, LAPACK_Z_SELECT2 selctg, armpl_int_t n,
                          armpl_doublecomplex_t *a, armpl_int_t lda,
                          armpl_doublecomplex_t *b, armpl_int_t ldb,
                          armpl_int_t *sdim, armpl_doublecomplex_t *alpha,
                          armpl_doublecomplex_t *beta,
                          armpl_doublecomplex_t *vsl, armpl_int_t ldvsl,
                          armpl_doublecomplex_t *vsr, armpl_int_t ldvsr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgges*, *LAPACKE_dgges*, *LAPACKE_sgges* and *LAPACKE_zgges*. It also exists with a native Fortran interface as *zgges*.

4.19.1935 LAPACKE_zgges3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgges3(armpl_int_t matrix_layout, char jobvsl,
                           char jobvsr, char sort, LAPACK_Z_SELECT2 selctg,
                           armpl_int_t n, armpl_doublecomplex_t *a,
                           armpl_int_t lda, armpl_doublecomplex_t *b,
                           armpl_int_t ldb, armpl_int_t *sdim,
                           armpl_doublecomplex_t *alpha,
                           armpl_doublecomplex_t *beta,
                           armpl_doublecomplex_t *vsl, armpl_int_t ldvsl,
                           armpl_doublecomplex_t *vsr, armpl_int_t ldvsr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgges3*, *LAPACKE_dgges3*, *LAPACKE_sgges3* and *LAPACKE_zgges3*. It also exists with a native Fortran interface as *zgges3*.

4.19.1936 LAPACKE_zgges3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgges3_work(armpl_int_t matrix_layout, char jobvsl,
                                char jobvsr, char sort,
                                LAPACK_Z_SELECT2 selctg, armpl_int_t n,
                                armpl_doublecomplex_t *a, armpl_int_t lda,
                                armpl_doublecomplex_t *b, armpl_int_t ldb,
                                armpl_int_t *sdim,
                                armpl_doublecomplex_t *alpha,
                                armpl_doublecomplex_t *beta,
                                armpl_doublecomplex_t *vsl, armpl_int_t ldvsl,
                                armpl_doublecomplex_t *vsr, armpl_int_t ldvsr,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork, double *rwork,
                                armpl_int_t *bwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgges3_work*, *LAPACKE_dgges3_work*, *LAPACKE_sgges3_work* and *LAPACKE_zgges3_work*.

4.19.1937 LAPACKE_zgges_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgges_work(armpl_int_t matrix_layout, char jobvsl,
                               char jobvsr, char sort,
                               LAPACK_Z_SELECT2 selctg, armpl_int_t n,
                               armpl_doublecomplex_t *a, armpl_int_t lda,
                               armpl_doublecomplex_t *b, armpl_int_t ldb,
                               armpl_int_t *sdim,
                               armpl_doublecomplex_t *alpha,
                               armpl_doublecomplex_t *beta,
                               armpl_doublecomplex_t *vsl, armpl_int_t ldvsl,
                               armpl_doublecomplex_t *vsr, armpl_int_t ldvsr,
                               armpl_doublecomplex_t *work, armpl_int_t lwork,
                               double *rwork, armpl_int_t *bwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgges_work*, *LAPACKE_dgges_work*, *LAPACKE_sgges_work* and *LAPACKE_zgges_work*.

4.19.1938 LAPACKE_zggesx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zggesx(armpl_int_t matrix_layout, char jobvsl,
                           char jobvsr, char sort, LAPACK_Z_SELECT2 selctg,
                           char sense, armpl_int_t n,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           armpl_doublecomplex_t *b, armpl_int_t ldb,
                           armpl_int_t *sdim, armpl_doublecomplex_t *alpha,
                           armpl_doublecomplex_t *beta,
                           armpl_doublecomplex_t *vsl, armpl_int_t ldvsl,
                           armpl_doublecomplex_t *vsr, armpl_int_t ldvsr,
                           double *rconde, double *rcondv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggesx*, *LAPACKE_dggesx*, *LAPACKE_sggesx* and *LAPACKE_zggesx*. It also exists with a native Fortran interface as *zggesx*.

4.19.1939 LAPACKE_zggesx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zggesx_work(armpl_int_t matrix_layout, char jobvsl,
                                char jobvsr, char sort,
                                LAPACK_Z_SELECT2 selctg, char sense,
                                armpl_int_t n, armpl_doublecomplex_t *a,
                                armpl_int_t lda, armpl_doublecomplex_t *b,
                                armpl_int_t ldb, armpl_int_t *sdim,
                                armpl_doublecomplex_t *alpha,
                                armpl_doublecomplex_t *beta,
                                armpl_doublecomplex_t *vsl, armpl_int_t ldvsl,
                                armpl_doublecomplex_t *vsr, armpl_int_t ldvsr,
                                double *rconde, double *rcondv,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork, double *rwork,
                                armpl_int_t *iwork, armpl_int_t liwork,
                                armpl_int_t *bwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggesx_work*, *LAPACKE_dggesx_work*, *LAPACKE_sggesx_work* and *LAPACKE_zggesx_work*.

4.19.1940 LAPACKE_zggev

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zggev(armpl_int_t matrix_layout, char jobvl, char jobvr,
                          armpl_int_t n, armpl_doublecomplex_t *a,
                          armpl_int_t lda, armpl_doublecomplex_t *b,
                          armpl_int_t ldb, armpl_doublecomplex_t *alpha,
                          armpl_doublecomplex_t *beta,
                          armpl_doublecomplex_t *vl, armpl_int_t ldvl,
                          armpl_doublecomplex_t *vr, armpl_int_t ldvr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggev*, *LAPACKE_dggev*, *LAPACKE_sggev* and *LAPACKE_zggev*. It also exists with a native Fortran interface as *zggev*.

4.19.1941 LAPACKE_zggev3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zggev3(armpl_int_t matrix_layout, char jobvl, char jobvr,
                           armpl_int_t n, armpl_doublecomplex_t *a,
                           armpl_int_t lda, armpl_doublecomplex_t *b,
                           armpl_int_t ldb, armpl_doublecomplex_t *alpha,
                           armpl_doublecomplex_t *beta,
                           armpl_doublecomplex_t *vl, armpl_int_t ldvl,
                           armpl_doublecomplex_t *vr, armpl_int_t ldvr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggev3*, *LAPACKE_dggev3*, *LAPACKE_sggev3* and *LAPACKE_zggev3*. It also exists with a native Fortran interface as *zggev3*.

4.19.1942 LAPACKE_zggev3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zggev3_work(armpl_int_t matrix_layout, char jobvl,
                                char jobvr, armpl_int_t n,
                                armpl_doublecomplex_t *a, armpl_int_t lda,
                                armpl_doublecomplex_t *b, armpl_int_t ldb,
                                armpl_doublecomplex_t *alpha,
                                armpl_doublecomplex_t *beta,
                                armpl_doublecomplex_t *vl, armpl_int_t ldvl,
                                armpl_doublecomplex_t *vr, armpl_int_t ldvr,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggev3_work*, *LAPACKE_dggev3_work*, *LAPACKE_sggev3_work* and *LAPACKE_zggev3_work*.

4.19.1943 LAPACKE_zggev_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zggev_work(armpl_int_t matrix_layout, char jobvl,
                               char jobvr, armpl_int_t n,
                               armpl_doublecomplex_t *a, armpl_int_t lda,
                               armpl_doublecomplex_t *b, armpl_int_t ldb,
                               armpl_doublecomplex_t *alpha,
                               armpl_doublecomplex_t *beta,
                               armpl_doublecomplex_t *vl, armpl_int_t ldvl,
                               armpl_doublecomplex_t *vr, armpl_int_t ldvr,
                               armpl_doublecomplex_t *work, armpl_int_t lwork,
                               double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggev_work*, *LAPACKE_dggev_work*, *LAPACKE_sggev_work* and *LAPACKE_zggev_work*.

4.19.1944 LAPACKE_zggevx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zggevx(armpl_int_t matrix_layout, char balanc, char jobvl,
                           char jobvr, char sense, armpl_int_t n,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           armpl_doublecomplex_t *b, armpl_int_t ldb,
                           armpl_doublecomplex_t *alpha,
                           armpl_doublecomplex_t *beta,
                           armpl_doublecomplex_t *vl, armpl_int_t ldvl,
                           armpl_doublecomplex_t *vr, armpl_int_t ldvr,
                           armpl_int_t *ilo, armpl_int_t *ihi, double *lscale,
                           double *rscale, double *abnrm, double *bbnrm,
                           double *rconde, double *rcondv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggevx*, *LAPACKE_dggevx*, *LAPACKE_sggevx* and *LAPACKE_zggevx*. It also exists with a native Fortran interface as *zggevx*.

4.19.1945 LAPACKE_zggevx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zggevx_work(armpl_int_t matrix_layout, char balanc,
                                char jobvl, char jobvr, char sense,
                                armpl_int_t n, armpl_doublecomplex_t *a,
                                armpl_int_t lda, armpl_doublecomplex_t *b,
                                armpl_int_t ldb, armpl_doublecomplex_t *alpha,
                                armpl_doublecomplex_t *beta,
                                armpl_doublecomplex_t *vl, armpl_int_t ldvl,
                                armpl_doublecomplex_t *vr, armpl_int_t ldvr,
                                armpl_int_t *ilo, armpl_int_t *ihi,
                                double *lscale, double *rscale, double *abnrm,
                                double *bbnrm, double *rconde, double *rcondv,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork, double *rwork,
                                armpl_int_t *iwork, armpl_int_t *bwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggevx_work*, *LAPACKE_dggevx_work*, *LAPACKE_sggevx_work* and *LAPACKE_zggevx_work*.

4.19.1946 LAPACKE_zggglm

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zggglm(armpl_int_t matrix_layout, armpl_int_t n,
                           armpl_int_t m, armpl_int_t p,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           armpl_doublecomplex_t *b, armpl_int_t ldb,
                           armpl_doublecomplex_t *d, armpl_doublecomplex_t *x,
                           armpl_doublecomplex_t *y);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggglm*, *LAPACKE_dggglm*, *LAPACKE_sggglm* and *LAPACKE_zggglm*. It also exists with a native Fortran interface as *zggglm*.

4.19.1947 LAPACKE_zggglm_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zggglm_work(armpl_int_t matrix_layout, armpl_int_t n,
                                armpl_int_t m, armpl_int_t p,
                                armpl_doublecomplex_t *a, armpl_int_t lda,
                                armpl_doublecomplex_t *b, armpl_int_t ldb,
                                armpl_doublecomplex_t *d,
                                armpl_doublecomplex_t *x,
                                armpl_doublecomplex_t *y,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggglm_work*, *LAPACKE_dggglm_work*, *LAPACKE_sggglm_work* and *LAPACKE_zggglm_work*.

4.19.1948 LAPACKE_zgghd3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgghd3(armpl_int_t matrix_layout, char compq, char compz,
                           armpl_int_t n, armpl_int_t ilo, armpl_int_t ihi,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           armpl_doublecomplex_t *b, armpl_int_t ldb,
                           armpl_doublecomplex_t *q, armpl_int_t ldq,
                           armpl_doublecomplex_t *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgghd3*, *LAPACKE_dgghd3*, *LAPACKE_sgghd3* and *LAPACKE_zgghd3*. It also exists with a native Fortran interface as *zgghd3*.

4.19.1949 LAPACKE_zgghd3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgghd3_work(armpl_int_t matrix_layout, char compq,
                                char compz, armpl_int_t n, armpl_int_t ilo,
                                armpl_int_t ihi, armpl_doublecomplex_t *a,
                                armpl_int_t lda, armpl_doublecomplex_t *b,
                                armpl_int_t ldb, armpl_doublecomplex_t *q,
                                armpl_int_t ldq, armpl_doublecomplex_t *z,
                                armpl_int_t ldz, armpl_doublecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgghd3_work*, *LAPACKE_dgghd3_work*, *LAPACKE_sgghd3_work* and *LAPACKE_zgghd3_work*.

4.19.1950 LAPACKE_zgghrd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgghrd(armpl_int_t matrix_layout, char compq, char compz,
                           armpl_int_t n, armpl_int_t ilo, armpl_int_t ihi,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           armpl_doublecomplex_t *b, armpl_int_t ldb,
                           armpl_doublecomplex_t *q, armpl_int_t ldq,
                           armpl_doublecomplex_t *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgghrd*, *LAPACKE_dgghrd*, *LAPACKE_sgghrd* and *LAPACKE_zgghrd*. It also exists with a native Fortran interface as *zgghrd*.

4.19.1951 LAPACKE_zgghrd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgghrd_work(armpl_int_t matrix_layout, char compq,
                                char compz, armpl_int_t n, armpl_int_t ilo,
                                armpl_int_t ihi, armpl_doublecomplex_t *a,
                                armpl_int_t lda, armpl_doublecomplex_t *b,
                                armpl_int_t ldb, armpl_doublecomplex_t *q,
                                armpl_int_t ldq, armpl_doublecomplex_t *z,
                                armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgghrd_work*, *LAPACKE_dgghrd_work*, *LAPACKE_sgghrd_work* and *LAPACKE_zgghrd_work*.

4.19.1952 LAPACKE_zgglse

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgglse(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t p,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           armpl_doublecomplex_t *b, armpl_int_t ldb,
                           armpl_doublecomplex_t *c, armpl_doublecomplex_t *d,
                           armpl_doublecomplex_t *x);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgglse*, *LAPACKE_dgglse*, *LAPACKE_sgglse* and *LAPACKE_zgglse*. It also exists with a native Fortran interface as *zgglse*.

4.19.1953 LAPACKE_zgglse_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgglse_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t p,
                                armpl_doublecomplex_t *a, armpl_int_t lda,
                                armpl_doublecomplex_t *b, armpl_int_t ldb,
                                armpl_doublecomplex_t *c,
                                armpl_doublecomplex_t *d,
                                armpl_doublecomplex_t *x,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgglse_work*, *LAPACKE_dgglse_work*, *LAPACKE_sgglse_work* and *LAPACKE_zgglse_work*.

4.19.1954 LAPACKE_zggqrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zggqrf(armpl_int_t matrix_layout, armpl_int_t n,
                           armpl_int_t m, armpl_int_t p,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           armpl_doublecomplex_t *taua,
                           armpl_doublecomplex_t *b, armpl_int_t ldb,
                           armpl_doublecomplex_t *taub);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggqrf*, *LAPACKE_dggqrf*, *LAPACKE_sggqrf* and *LAPACKE_zggqrf*. It also exists with a native Fortran interface as *zggqrf*.

4.19.1955 LAPACKE_zggqrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zggqrf_work(armpl_int_t matrix_layout, armpl_int_t n,
                                armpl_int_t m, armpl_int_t p,
                                armpl_doublecomplex_t *a, armpl_int_t lda,
                                armpl_doublecomplex_t *taua,
                                armpl_doublecomplex_t *b, armpl_int_t ldb,
                                armpl_doublecomplex_t *taub,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggqrf_work*, *LAPACKE_dggqrf_work*, *LAPACKE_sggqrf_work* and *LAPACKE_zggqrf_work*.

4.19.1956 LAPACKE_zggrqf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zggrqf(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t p, armpl_int_t n,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           armpl_doublecomplex_t *taua,
                           armpl_doublecomplex_t *b, armpl_int_t ldb,
                           armpl_doublecomplex_t *taub);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggrqf*, *LAPACKE_dggrqf*, *LAPACKE_sggrqf* and *LAPACKE_zggrqf*. It also exists with a native Fortran interface as *zggrqf*.

4.19.1957 LAPACKE_zggrqf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zggrqf_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t p, armpl_int_t n,
                                armpl_doublecomplex_t *a, armpl_int_t lda,
                                armpl_doublecomplex_t *taua,
                                armpl_doublecomplex_t *b, armpl_int_t ldb,
                                armpl_doublecomplex_t *taub,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggrqf_work*, *LAPACKE_dggrqf_work*, *LAPACKE_sggrqf_work* and *LAPACKE_zggrqf_work*.

4.19.1958 LAPACKE_zggsvd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zggsvd(armpl_int_t matrix_layout, char jobu, char jobv,
                           char jobq, armpl_int_t m, armpl_int_t n,
                           armpl_int_t p, armpl_int_t *k, armpl_int_t *l,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           armpl_doublecomplex_t *b, armpl_int_t ldb,
                           double *alpha, double *beta,
                           armpl_doublecomplex_t *u, armpl_int_t ldu,
                           armpl_doublecomplex_t *v, armpl_int_t ldv,
                           armpl_doublecomplex_t *q, armpl_int_t ldq,
                           armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggsvd*, *LAPACKE_dggsvd*, *LAPACKE_sggsvd* and *LAPACKE_zggsvd*.

4.19.1959 LAPACKE_zggsvd3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zggsvd3(armpl_int_t matrix_layout, char jobu, char jobv,
                            char jobq, armpl_int_t m, armpl_int_t n,
                            armpl_int_t p, armpl_int_t *k, armpl_int_t *l,
                            armpl_doublecomplex_t *a, armpl_int_t lda,
                            armpl_doublecomplex_t *b, armpl_int_t ldb,
                            double *alpha, double *beta,
                            armpl_doublecomplex_t *u, armpl_int_t ldu,
                            armpl_doublecomplex_t *v, armpl_int_t ldv,
                            armpl_doublecomplex_t *q, armpl_int_t ldq,
                            armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggsvd3*, *LAPACKE_dggsvd3*, *LAPACKE_sggsvd3* and *LAPACKE_zggsvd3*. It also exists with a native Fortran interface as *zggsvd3*.

4.19.1960 LAPACKE_zggsvd3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zggsvd3_work(armpl_int_t matrix_layout, char jobu,
                                 char jobv, char jobq, armpl_int_t m,
                                 armpl_int_t n, armpl_int_t p, armpl_int_t *k,
                                 armpl_int_t *l, armpl_doublecomplex_t *a,
                                 armpl_int_t lda, armpl_doublecomplex_t *b,
                                 armpl_int_t ldb, double *alpha, double *beta,
                                 armpl_doublecomplex_t *u, armpl_int_t ldu,
                                 armpl_doublecomplex_t *v, armpl_int_t ldv,
                                 armpl_doublecomplex_t *q, armpl_int_t ldq,
                                 armpl_doublecomplex_t *work,
                                 armpl_int_t lwork, double *rwork,
                                 armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggsvd3_work*, *LAPACKE_dggsvd3_work*, *LAPACKE_sggsvd3_work* and *LAPACKE_zggsvd3_work*.

4.19.1961 LAPACKE_zggsvd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zggsvd_work(armpl_int_t matrix_layout, char jobu,
                                char jobv, char jobq, armpl_int_t m,
                                armpl_int_t n, armpl_int_t p, armpl_int_t *k,
                                armpl_int_t *l, armpl_doublecomplex_t *a,
                                armpl_int_t lda, armpl_doublecomplex_t *b,
                                armpl_int_t ldb, double *alpha, double *beta,
                                armpl_doublecomplex_t *u, armpl_int_t ldu,
                                armpl_doublecomplex_t *v, armpl_int_t ldv,
                                armpl_doublecomplex_t *q, armpl_int_t ldq,
                                armpl_doublecomplex_t *work, double *rwork,
                                armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggsvd_work*, *LAPACKE_dggsvd_work*, *LAPACKE_sggsvd_work* and *LAPACKE_zggsvd_work*.

4.19.1962 LAPACKE_zggsvp

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zggsvp(armpl_int_t matrix_layout, char jobu, char jobv,
                           char jobq, armpl_int_t m, armpl_int_t p,
                           armpl_int_t n, armpl_doublecomplex_t *a,
                           armpl_int_t lda, armpl_doublecomplex_t *b,
                           armpl_int_t ldb, double tola, double tolb,
```

(continues on next page)

(continued from previous page)

```
                            armpl_int_t *k, armpl_int_t *l,
                            armpl_doublecomplex_t *u, armpl_int_t ldu,
                            armpl_doublecomplex_t *v, armpl_int_t ldv,
                            armpl_doublecomplex_t *q, armpl_int_t ldq);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggsvp*, *LAPACKE_dggsvp*, *LAPACKE_sggsvp* and *LAPACKE_zggsvp*.

4.19.1963 LAPACKE_zggsvp3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zggsvp3(armpl_int_t matrix_layout, char jobu, char jobv,
                            char jobq, armpl_int_t m, armpl_int_t p,
                            armpl_int_t n, armpl_doublecomplex_t *a,
                            armpl_int_t lda, armpl_doublecomplex_t *b,
                            armpl_int_t ldb, double tola, double tolb,
                            armpl_int_t *k, armpl_int_t *l,
                            armpl_doublecomplex_t *u, armpl_int_t ldu,
                            armpl_doublecomplex_t *v, armpl_int_t ldv,
                            armpl_doublecomplex_t *q, armpl_int_t ldq);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggsvp3*, *LAPACKE_dggsvp3*, *LAPACKE_sggsvp3* and *LAPACKE_zggsvp3*. It also exists with a native Fortran interface as *zggsvp3*.

4.19.1964 LAPACKE_zggsvp3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zggsvp3_work(armpl_int_t matrix_layout, char jobu,
                                 char jobv, char jobq, armpl_int_t m,
                                 armpl_int_t p, armpl_int_t n,
                                 armpl_doublecomplex_t *a, armpl_int_t lda,
                                 armpl_doublecomplex_t *b, armpl_int_t ldb,
                                 double tola, double tolb, armpl_int_t *k,
                                 armpl_int_t *l, armpl_doublecomplex_t *u,
                                 armpl_int_t ldu, armpl_doublecomplex_t *v,
                                 armpl_int_t ldv, armpl_doublecomplex_t *q,
                                 armpl_int_t ldq, armpl_int_t *iwork,
                                 double *rwork, armpl_doublecomplex_t *tau,
                                 armpl_doublecomplex_t *work,
                                 armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggsvp3_work*, *LAPACKE_dggsvp3_work*, *LAPACKE_sggsvp3_work* and *LAPACKE_zggsvp3_work*.

4.19.1965 LAPACKE_zggsvp_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zggsvp_work(armpl_int_t matrix_layout, char jobu,
                                char jobv, char jobq, armpl_int_t m,
                                armpl_int_t p, armpl_int_t n,
                                armpl_doublecomplex_t *a, armpl_int_t lda,
                                armpl_doublecomplex_t *b, armpl_int_t ldb,
                                double tola, double tolb, armpl_int_t *k,
                                armpl_int_t *l, armpl_doublecomplex_t *u,
                                armpl_int_t ldu, armpl_doublecomplex_t *v,
                                armpl_int_t ldv, armpl_doublecomplex_t *q,
                                armpl_int_t ldq, armpl_int_t *iwork,
                                double *rwork, armpl_doublecomplex_t *tau,
                                armpl_doublecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cggsvp_work*, *LAPACKE_dggsvp_work*, *LAPACKE_sggsvp_work* and *LAPACKE_zggsvp_work*.

4.19.1966 LAPACKE_zgtcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgtcon(char norm, armpl_int_t n,
                           const armpl_doublecomplex_t *dl,
                           const armpl_doublecomplex_t *d,
                           const armpl_doublecomplex_t *du,
                           const armpl_doublecomplex_t *du2,
                           const armpl_int_t *ipiv, double anorm,
                           double *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgtcon*, *LAPACKE_dgtcon*, *LAPACKE_sgtcon* and *LAPACKE_zgtcon*. It also exists with a native Fortran interface as *zgtcon*.

4.19.1967 LAPACKE_zgtcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgtcon_work(char norm, armpl_int_t n,
                                const armpl_doublecomplex_t *dl,
```

(continues on next page)

(continued from previous page)

```
                                const armpl_doublecomplex_t *d,
                                const armpl_doublecomplex_t *du,
                                const armpl_doublecomplex_t *du2,
                                const armpl_int_t *ipiv, double anorm,
                                double *rcond, armpl_doublecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgtcon_work*, *LAPACKE_dgtcon_work*, *LAPACKE_sgtcon_work* and *LAPACKE_zgtcon_work*.

4.19.1968 LAPACKE_zgtrfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgtrfs(armpl_int_t matrix_layout, char trans,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_doublecomplex_t *dl,
                           const armpl_doublecomplex_t *d,
                           const armpl_doublecomplex_t *du,
                           const armpl_doublecomplex_t *dlf,
                           const armpl_doublecomplex_t *df,
                           const armpl_doublecomplex_t *duf,
                           const armpl_doublecomplex_t *du2,
                           const armpl_int_t *ipiv,
                           const armpl_doublecomplex_t *b, armpl_int_t ldb,
                           armpl_doublecomplex_t *x, armpl_int_t ldx,
                           double *ferr, double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgtrfs*, *LAPACKE_dgtrfs*, *LAPACKE_sgtrfs* and *LAPACKE_zgtrfs*. It also exists with a native Fortran interface as *zgtrfs*.

4.19.1969 LAPACKE_zgtrfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgtrfs_work(armpl_int_t matrix_layout, char trans,
                                armpl_int_t n, armpl_int_t nrhs,
                                const armpl_doublecomplex_t *dl,
                                const armpl_doublecomplex_t *d,
                                const armpl_doublecomplex_t *du,
                                const armpl_doublecomplex_t *dlf,
                                const armpl_doublecomplex_t *df,
                                const armpl_doublecomplex_t *duf,
                                const armpl_doublecomplex_t *du2,
                                const armpl_int_t *ipiv,
                                const armpl_doublecomplex_t *b,
                                armpl_int_t ldb, armpl_doublecomplex_t *x,
                                armpl_int_t ldx, double *ferr, double *berr,
                                armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgtrfs_work*, *LAPACKE_dgtrfs_work*, *LAPACKE_sgtrfs_work* and *LAPACKE_zgtrfs_work*.

4.19.1970 LAPACKE_zgtsv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgtsv(armpl_int_t matrix_layout, armpl_int_t n,
                          armpl_int_t nrhs, armpl_doublecomplex_t *dl,
                          armpl_doublecomplex_t *d, armpl_doublecomplex_t *du,
                          armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgtsv*, *LAPACKE_dgtsv*, *LAPACKE_sgtsv* and *LAPACKE_zgtsv*. It also exists with a native Fortran interface as *zgtsv*.

4.19.1971 LAPACKE_zgtsv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgtsv_work(armpl_int_t matrix_layout, armpl_int_t n,
                               armpl_int_t nrhs, armpl_doublecomplex_t *dl,
                               armpl_doublecomplex_t *d,
                               armpl_doublecomplex_t *du,
                               armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgtsv_work*, *LAPACKE_dgtsv_work*, *LAPACKE_sgtsv_work* and *LAPACKE_zgtsv_work*.

4.19.1972 LAPACKE_zgtsvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgtsvx(armpl_int_t matrix_layout, char fact, char trans,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_doublecomplex_t *dl,
                           const armpl_doublecomplex_t *d,
```

(continues on next page)

(continued from previous page)

```
                                const armpl_doublecomplex_t *du,
                                armpl_doublecomplex_t *dlf,
                                armpl_doublecomplex_t *df,
                                armpl_doublecomplex_t *duf,
                                armpl_doublecomplex_t *du2, armpl_int_t *ipiv,
                                const armpl_doublecomplex_t *b, armpl_int_t ldb,
                                armpl_doublecomplex_t *x, armpl_int_t ldx,
                                double *rcond, double *ferr, double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgtsvx*, *LAPACKE_dgtsvx*, *LAPACKE_sgtsvx* and *LAPACKE_zgtsvx*. It also exists with a native Fortran interface as *zgtsvx*.

4.19.1973 LAPACKE_zgtsvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgtsvx_work(armpl_int_t matrix_layout, char fact,
                                char trans, armpl_int_t n, armpl_int_t nrhs,
                                const armpl_doublecomplex_t *dl,
                                const armpl_doublecomplex_t *d,
                                const armpl_doublecomplex_t *du,
                                armpl_doublecomplex_t *dlf,
                                armpl_doublecomplex_t *df,
                                armpl_doublecomplex_t *duf,
                                armpl_doublecomplex_t *du2, armpl_int_t *ipiv,
                                const armpl_doublecomplex_t *b,
                                armpl_int_t ldb, armpl_doublecomplex_t *x,
                                armpl_int_t ldx, double *rcond, double *ferr,
                                double *berr, armpl_doublecomplex_t *work,
                                double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgtsvx_work*, *LAPACKE_dgtsvx_work*, *LAPACKE_sgtsvx_work* and *LAPACKE_zgtsvx_work*.

4.19.1974 LAPACKE_zgttrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgttrf(armpl_int_t n, armpl_doublecomplex_t *dl,
                           armpl_doublecomplex_t *d,
                           armpl_doublecomplex_t *du,
                           armpl_doublecomplex_t *du2, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgttrf*, *LAPACKE_dgttrf*, *LAPACKE_sgttrf* and *LAPACKE_zgttrf*. It also exists with a native Fortran interface as *zgttrf*.

4.19.1975 LAPACKE_zgttrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgttrf_work(armpl_int_t n, armpl_doublecomplex_t *dl,
                                armpl_doublecomplex_t *d,
                                armpl_doublecomplex_t *du,
                                armpl_doublecomplex_t *du2,
                                armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgttrf_work*, *LAPACKE_dgttrf_work*, *LAPACKE_sgttrf_work* and *LAPACKE_zgttrf_work*.

4.19.1976 LAPACKE_zgttrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgttrs(armpl_int_t matrix_layout, char trans,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_doublecomplex_t *dl,
                           const armpl_doublecomplex_t *d,
                           const armpl_doublecomplex_t *du,
                           const armpl_doublecomplex_t *du2,
                           const armpl_int_t *ipiv, armpl_doublecomplex_t *b,
                           armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgttrs*, *LAPACKE_dgttrs*, *LAPACKE_sgttrs* and *LAPACKE_zgttrs*. It also exists with a native Fortran interface as *zgttrs*.

4.19.1977 LAPACKE_zgttrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zgttrs_work(armpl_int_t matrix_layout, char trans,
                                armpl_int_t n, armpl_int_t nrhs,
                                const armpl_doublecomplex_t *dl,
                                const armpl_doublecomplex_t *d,
                                const armpl_doublecomplex_t *du,
                                const armpl_doublecomplex_t *du2,
                                const armpl_int_t *ipiv,
                                armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cgttrs_work*, *LAPACKE_dgttrs_work*, *LAPACKE_sgttrs_work* and *LAPACKE_zgttrs_work*.

4.19.1978 LAPACKE_zhbev

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhbev(armpl_int_t matrix_layout, char jobz, char uplo,
                          armpl_int_t n, armpl_int_t kd,
                          armpl_doublecomplex_t *ab, armpl_int_t ldab,
                          double *w, armpl_doublecomplex_t *z,
                          armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbev* and *LAPACKE_zhbev*. It also exists with a native Fortran interface as *zhbev*.

4.19.1979 LAPACKE_zhbev_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhbev_2stage(armpl_int_t matrix_layout, char jobz,
                                 char uplo, armpl_int_t n, armpl_int_t kd,
                                 armpl_doublecomplex_t *ab, armpl_int_t ldab,
                                 double *w, armpl_doublecomplex_t *z,
                                 armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbev_2stage* and *LAPACKE_zhbev_2stage*. It also exists with a native Fortran interface as *zhbev_2stage*.

4.19.1980 LAPACKE_zhbev_2stage_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhbev_2stage_work(armpl_int_t matrix_layout, char jobz,
                                      char uplo, armpl_int_t n,
                                      armpl_int_t kd,
                                      armpl_doublecomplex_t *ab,
                                      armpl_int_t ldab, double *w,
                                      armpl_doublecomplex_t *z,
                                      armpl_int_t ldz,
                                      armpl_doublecomplex_t *work,
                                      armpl_int_t lwork, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbev_2stage_work* and *LAPACKE_zhbev_2stage_work*.

4.19.1981 LAPACKE_zhbev_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhbev_work(armpl_int_t matrix_layout, char jobz,
                               char uplo, armpl_int_t n, armpl_int_t kd,
                               armpl_doublecomplex_t *ab, armpl_int_t ldab,
                               double *w, armpl_doublecomplex_t *z,
                               armpl_int_t ldz, armpl_doublecomplex_t *work,
                               double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbev_work* and *LAPACKE_zhbev_work*.

4.19.1982 LAPACKE_zhbevd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhbevd(armpl_int_t matrix_layout, char jobz, char uplo,
                           armpl_int_t n, armpl_int_t kd,
                           armpl_doublecomplex_t *ab, armpl_int_t ldab,
                           double *w, armpl_doublecomplex_t *z,
                           armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbevd* and *LAPACKE_zhbevd*. It also exists with a native Fortran interface as *zhbevd*.

4.19.1983 LAPACKE_zhbevd_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhbevd_2stage(armpl_int_t matrix_layout, char jobz,
                                  char uplo, armpl_int_t n, armpl_int_t kd,
                                  armpl_doublecomplex_t *ab, armpl_int_t ldab,
                                  double *w, armpl_doublecomplex_t *z,
                                  armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbevd_2stage* and *LAPACKE_zhbevd_2stage*. It also exists with a native Fortran interface as *zhbevd_2stage*.

4.19.1984 LAPACKE_zhbevd_2stage_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhbevd_2stage_work(armpl_int_t matrix_layout, char jobz,
                                       char uplo, armpl_int_t n,
                                       armpl_int_t kd,
                                       armpl_doublecomplex_t *ab,
                                       armpl_int_t ldab, double *w,
                                       armpl_doublecomplex_t *z,
                                       armpl_int_t ldz,
                                       armpl_doublecomplex_t *work,
                                       armpl_int_t lwork, double *rwork,
                                       armpl_int_t lrwork, armpl_int_t *iwork,
                                       armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbevd_2stage_work* and *LAPACKE_zhbevd_2stage_work*.

4.19.1985 LAPACKE_zhbevd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhbevd_work(armpl_int_t matrix_layout, char jobz,
                                char uplo, armpl_int_t n, armpl_int_t kd,
                                armpl_doublecomplex_t *ab, armpl_int_t ldab,
                                double *w, armpl_doublecomplex_t *z,
                                armpl_int_t ldz, armpl_doublecomplex_t *work,
                                armpl_int_t lwork, double *rwork,
                                armpl_int_t lrwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbevd_work* and *LAPACKE_zhbevd_work*.

4.19.1986 LAPACKE_zhbevx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhbevx(armpl_int_t matrix_layout, char jobz, char range,
                           char uplo, armpl_int_t n, armpl_int_t kd,
                           armpl_doublecomplex_t *ab, armpl_int_t ldab,
                           armpl_doublecomplex_t *q, armpl_int_t ldq,
                           double vl, double vu, armpl_int_t il,
                           armpl_int_t iu, double abstol, armpl_int_t *m,
                           double *w, armpl_doublecomplex_t *z,
                           armpl_int_t ldz, armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbevx* and *LAPACKE_zhbevx*. It also exists with a native Fortran interface as *zhbevx*.

4.19.1987 LAPACKE_zhbevx_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhbevx_2stage(armpl_int_t matrix_layout, char jobz,
                                  char range, char uplo, armpl_int_t n,
                                  armpl_int_t kd, armpl_doublecomplex_t *ab,
                                  armpl_int_t ldab, armpl_doublecomplex_t *q,
                                  armpl_int_t ldq, double vl, double vu,
                                  armpl_int_t il, armpl_int_t iu,
                                  double abstol, armpl_int_t *m, double *w,
                                  armpl_doublecomplex_t *z, armpl_int_t ldz,
                                  armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbevx_2stage* and *LAPACKE_zhbevx_2stage*. It also exists with a native Fortran interface as *zhbevx_2stage*.

4.19.1988 LAPACKE_zhbevx_2stage_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhbevx_2stage_work(armpl_int_t matrix_layout, char jobz,
                                       char range, char uplo, armpl_int_t n,
                                       armpl_int_t kd,
                                       armpl_doublecomplex_t *ab,
                                       armpl_int_t ldab,
                                       armpl_doublecomplex_t *q,
```

(continues on next page)

(continued from previous page)

```
                                        armpl_int_t ldq, double vl, double vu,
                                        armpl_int_t il, armpl_int_t iu,
                                        double abstol, armpl_int_t *m,
                                        double *w, armpl_doublecomplex_t *z,
                                        armpl_int_t ldz,
                                        armpl_doublecomplex_t *work,
                                        armpl_int_t lwork, double *rwork,
                                        armpl_int_t *iwork,
                                        armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbevx_2stage_work* and *LAPACKE_zhbevx_2stage_work*.

4.19.1989 LAPACKE_zhbevx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhbevx_work(armpl_int_t matrix_layout, char jobz,
                                char range, char uplo, armpl_int_t n,
                                armpl_int_t kd, armpl_doublecomplex_t *ab,
                                armpl_int_t ldab, armpl_doublecomplex_t *q,
                                armpl_int_t ldq, double vl, double vu,
                                armpl_int_t il, armpl_int_t iu, double abstol,
                                armpl_int_t *m, double *w,
                                armpl_doublecomplex_t *z, armpl_int_t ldz,
                                armpl_doublecomplex_t *work, double *rwork,
                                armpl_int_t *iwork, armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbevx_work* and *LAPACKE_zhbevx_work*.

4.19.1990 LAPACKE_zhbgst

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhbgst(armpl_int_t matrix_layout, char vect, char uplo,
                           armpl_int_t n, armpl_int_t ka, armpl_int_t kb,
                           armpl_doublecomplex_t *ab, armpl_int_t ldab,
                           const armpl_doublecomplex_t *bb, armpl_int_t ldbb,
                           armpl_doublecomplex_t *x, armpl_int_t ldx);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbgst* and *LAPACKE_zhbgst*. It also exists with a native Fortran interface as *zhbgst*.

4.19.1991 LAPACKE_zhbgst_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhbgst_work(armpl_int_t matrix_layout, char vect,
                                char uplo, armpl_int_t n, armpl_int_t ka,
                                armpl_int_t kb, armpl_doublecomplex_t *ab,
                                armpl_int_t ldab,
                                const armpl_doublecomplex_t *bb,
                                armpl_int_t ldbb, armpl_doublecomplex_t *x,
                                armpl_int_t ldx, armpl_doublecomplex_t *work,
                                double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbgst_work* and *LAPACKE_zhbgst_work*.

4.19.1992 LAPACKE_zhbgv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhbgv(armpl_int_t matrix_layout, char jobz, char uplo,
                          armpl_int_t n, armpl_int_t ka, armpl_int_t kb,
                          armpl_doublecomplex_t *ab, armpl_int_t ldab,
                          armpl_doublecomplex_t *bb, armpl_int_t ldbb,
                          double *w, armpl_doublecomplex_t *z,
                          armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbgv* and *LAPACKE_zhbgv*. It also exists with a native Fortran interface as *zhbgv*.

4.19.1993 LAPACKE_zhbgv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhbgv_work(armpl_int_t matrix_layout, char jobz,
                               char uplo, armpl_int_t n, armpl_int_t ka,
                               armpl_int_t kb, armpl_doublecomplex_t *ab,
                               armpl_int_t ldab, armpl_doublecomplex_t *bb,
                               armpl_int_t ldbb, double *w,
                               armpl_doublecomplex_t *z, armpl_int_t ldz,
                               armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbgv_work* and *LAPACKE_zhbgv_work*.

4.19.1994 LAPACKE_zhbgvd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhbgvd(armpl_int_t matrix_layout, char jobz, char uplo,
                           armpl_int_t n, armpl_int_t ka, armpl_int_t kb,
                           armpl_doublecomplex_t *ab, armpl_int_t ldab,
                           armpl_doublecomplex_t *bb, armpl_int_t ldbb,
                           double *w, armpl_doublecomplex_t *z,
                           armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbgvd* and *LAPACKE_zhbgvd*. It also exists with a native Fortran interface as *zhbgvd*.

4.19.1995 LAPACKE_zhbgvd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhbgvd_work(armpl_int_t matrix_layout, char jobz,
                                char uplo, armpl_int_t n, armpl_int_t ka,
                                armpl_int_t kb, armpl_doublecomplex_t *ab,
                                armpl_int_t ldab, armpl_doublecomplex_t *bb,
```

(continues on next page)

(continued from previous page)

```
                                armpl_int_t ldbb, double *w,
                                armpl_doublecomplex_t *z, armpl_int_t ldz,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork, double *rwork,
                                armpl_int_t lrwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbgvd_work* and *LAPACKE_zhbgvd_work*.

4.19.1996 LAPACKE_zhbgvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhbgvx(armpl_int_t matrix_layout, char jobz, char range,
                           char uplo, armpl_int_t n, armpl_int_t ka,
                           armpl_int_t kb, armpl_doublecomplex_t *ab,
                           armpl_int_t ldab, armpl_doublecomplex_t *bb,
                           armpl_int_t ldbb, armpl_doublecomplex_t *q,
                           armpl_int_t ldq, double vl, double vu,
                           armpl_int_t il, armpl_int_t iu, double abstol,
                           armpl_int_t *m, double *w,
                           armpl_doublecomplex_t *z, armpl_int_t ldz,
                           armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbgvx* and *LAPACKE_zhbgvx*. It also exists with a native Fortran interface as *zhbgvx*.

4.19.1997 LAPACKE_zhbgvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhbgvx_work(armpl_int_t matrix_layout, char jobz,
                                char range, char uplo, armpl_int_t n,
                                armpl_int_t ka, armpl_int_t kb,
                                armpl_doublecomplex_t *ab, armpl_int_t ldab,
                                armpl_doublecomplex_t *bb, armpl_int_t ldbb,
                                armpl_doublecomplex_t *q, armpl_int_t ldq,
                                double vl, double vu, armpl_int_t il,
                                armpl_int_t iu, double abstol, armpl_int_t *m,
                                double *w, armpl_doublecomplex_t *z,
                                armpl_int_t ldz, armpl_doublecomplex_t *work,
                                double *rwork, armpl_int_t *iwork,
                                armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbgvx_work* and *LAPACKE_zhbgvx_work*.

4.19.1998 LAPACKE_zhbtrd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhbtrd(armpl_int_t matrix_layout, char vect, char uplo,
                           armpl_int_t n, armpl_int_t kd,
                           armpl_doublecomplex_t *ab, armpl_int_t ldab,
                           double *d, double *e, armpl_doublecomplex_t *q,
                           armpl_int_t ldq);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbtrd* and *LAPACKE_zhbtrd*. It also exists with a native Fortran interface as *zhbtrd*.

4.19.1999 LAPACKE_zhbtrd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhbtrd_work(armpl_int_t matrix_layout, char vect,
                                char uplo, armpl_int_t n, armpl_int_t kd,
                                armpl_doublecomplex_t *ab, armpl_int_t ldab,
                                double *d, double *e,
                                armpl_doublecomplex_t *q, armpl_int_t ldq,
                                armpl_doublecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chbtrd_work* and *LAPACKE_zhbtrd_work*.

4.19.2000 LAPACKE_zhecon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhecon(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, const armpl_doublecomplex_t *a,
                           armpl_int_t lda, const armpl_int_t *ipiv,
                           double anorm, double *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_checon* and *LAPACKE_zhecon*. It also exists with a native Fortran interface as *zhecon*.

4.19.2001 LAPACKE_zhecon_3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhecon_3(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, const armpl_doublecomplex_t *a,
                             armpl_int_t lda, const armpl_doublecomplex_t *e,
                             const armpl_int_t *ipiv, double anorm,
                             double *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_checon_3* and *LAPACKE_zhecon_3*. It also exists with a native Fortran interface as *zhecon_3*.

4.19.2002 LAPACKE_zhecon_3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhecon_3_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n,
                                  const armpl_doublecomplex_t *a,
                                  armpl_int_t lda,
                                  const armpl_doublecomplex_t *e,
                                  const armpl_int_t *ipiv, double anorm,
                                  double *rcond,
                                  armpl_doublecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_checon_3_work* and *LAPACKE_zhecon_3_work*.

4.19.2003 LAPACKE_zhecon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhecon_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, const armpl_doublecomplex_t *a,
                                armpl_int_t lda, const armpl_int_t *ipiv,
                                double anorm, double *rcond,
                                armpl_doublecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_checon_work* and *LAPACKE_zhecon_work*.

4.19.2004 LAPACKE_zheequb

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zheequb(armpl_int_t matrix_layout, char uplo,
                            armpl_int_t n, const armpl_doublecomplex_t *a,
                            armpl_int_t lda, double *s, double *scond,
                            double *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.2005 LAPACKE_zheequb_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zheequb_work(armpl_int_t matrix_layout, char uplo,
                                 armpl_int_t n,
                                 const armpl_doublecomplex_t *a,
                                 armpl_int_t lda, double *s, double *scond,
                                 double *amax, armpl_doublecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheequb_work* and *LAPACKE_zheequb_work*.

4.19.2006 LAPACKE_zheev

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zheev(armpl_int_t matrix_layout, char jobz, char uplo,
                          armpl_int_t n, armpl_doublecomplex_t *a,
                          armpl_int_t lda, double *w);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheev* and *LAPACKE_zheev*. It also exists with a native Fortran interface as *zheev*.

4.19.2007 LAPACKE_zheev_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zheev_2stage(armpl_int_t matrix_layout, char jobz,
                                 char uplo, armpl_int_t n,
                                 armpl_doublecomplex_t *a, armpl_int_t lda,
                                 double *w);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheev_2stage* and *LAPACKE_zheev_2stage*. It also exists with a native Fortran interface as *zheev_2stage*.

4.19.2008 LAPACKE_zheev_2stage_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zheev_2stage_work(armpl_int_t matrix_layout, char jobz,
                                      char uplo, armpl_int_t n,
                                      armpl_doublecomplex_t *a,
                                      armpl_int_t lda, double *w,
                                      armpl_doublecomplex_t *work,
                                      armpl_int_t lwork, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheev_2stage_work* and *LAPACKE_zheev_2stage_work*.

4.19.2009 LAPACKE_zheev_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zheev_work(armpl_int_t matrix_layout, char jobz,
                               char uplo, armpl_int_t n,
                               armpl_doublecomplex_t *a, armpl_int_t lda,
                               double *w, armpl_doublecomplex_t *work,
                               armpl_int_t lwork, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheev_work* and *LAPACKE_zheev_work*.

4.19.2010 LAPACKE_zheevd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zheevd(armpl_int_t matrix_layout, char jobz, char uplo,
                           armpl_int_t n, armpl_doublecomplex_t *a,
                           armpl_int_t lda, double *w);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheevd* and *LAPACKE_zheevd*. It also exists with a native Fortran interface as *zheevd*.

4.19.2011 LAPACKE_zheevd_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zheevd_2stage(armpl_int_t matrix_layout, char jobz,
                                  char uplo, armpl_int_t n,
                                  armpl_doublecomplex_t *a, armpl_int_t lda,
                                  double *w);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheevd_2stage* and *LAPACKE_zheevd_2stage*. It also exists with a native Fortran interface as *zheevd_2stage*.

4.19.2012 LAPACKE_zheevd_2stage_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zheevd_2stage_work(armpl_int_t matrix_layout, char jobz,
                                       char uplo, armpl_int_t n,
                                       armpl_doublecomplex_t *a,
                                       armpl_int_t lda, double *w,
                                       armpl_doublecomplex_t *work,
                                       armpl_int_t lwork, double *rwork,
                                       armpl_int_t lrwork, armpl_int_t *iwork,
                                       armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheevd_2stage_work* and *LAPACKE_zheevd_2stage_work*.

4.19.2013 LAPACKE_zheevd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zheevd_work(armpl_int_t matrix_layout, char jobz,
                                char uplo, armpl_int_t n,
                                armpl_doublecomplex_t *a, armpl_int_t lda,
                                double *w, armpl_doublecomplex_t *work,
                                armpl_int_t lwork, double *rwork,
                                armpl_int_t lrwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheevd_work* and *LAPACKE_zheevd_work*.

4.19.2014 LAPACKE_zheevr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zheevr(armpl_int_t matrix_layout, char jobz, char range,
                           char uplo, armpl_int_t n, armpl_doublecomplex_t *a,
                           armpl_int_t lda, double vl, double vu,
                           armpl_int_t il, armpl_int_t iu, double abstol,
                           armpl_int_t *m, double *w,
                           armpl_doublecomplex_t *z, armpl_int_t ldz,
                           armpl_int_t *isuppz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheevr* and *LAPACKE_zheevr*. It also exists with a native Fortran interface as *zheevr*.

4.19.2015 LAPACKE_zheevr_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zheevr_2stage(armpl_int_t matrix_layout, char jobz,
                                  char range, char uplo, armpl_int_t n,
                                  armpl_doublecomplex_t *a, armpl_int_t lda,
                                  double vl, double vu, armpl_int_t il,
                                  armpl_int_t iu, double abstol,
                                  armpl_int_t *m, double *w,
                                  armpl_doublecomplex_t *z, armpl_int_t ldz,
                                  armpl_int_t *isuppz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheevr_2stage* and *LAPACKE_zheevr_2stage*. It also exists with a native Fortran interface as *zheevr_2stage*.

4.19.2016 LAPACKE_zheevr_2stage_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zheevr_2stage_work(armpl_int_t matrix_layout, char jobz,
                                       char range, char uplo, armpl_int_t n,
                                       armpl_doublecomplex_t *a,
                                       armpl_int_t lda, double vl, double vu,
                                       armpl_int_t il, armpl_int_t iu,
                                       double abstol, armpl_int_t *m,
                                       double *w, armpl_doublecomplex_t *z,
                                       armpl_int_t ldz, armpl_int_t *isuppz,
                                       armpl_doublecomplex_t *work,
                                       armpl_int_t lwork, double *rwork,
                                       armpl_int_t lrwork, armpl_int_t *iwork,
                                       armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheevr_2stage_work* and *LAPACKE_zheevr_2stage_work*.

4.19.2017 LAPACKE_zheevr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zheevr_work(armpl_int_t matrix_layout, char jobz,
                                char range, char uplo, armpl_int_t n,
                                armpl_doublecomplex_t *a, armpl_int_t lda,
                                double vl, double vu, armpl_int_t il,
                                armpl_int_t iu, double abstol, armpl_int_t *m,
                                double *w, armpl_doublecomplex_t *z,
                                armpl_int_t ldz, armpl_int_t *isuppz,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork, double *rwork,
                                armpl_int_t lrwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheevr_work* and *LAPACKE_zheevr_work*.

4.19.2018 LAPACKE_zheevx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zheevx(armpl_int_t matrix_layout, char jobz, char range,
                           char uplo, armpl_int_t n, armpl_doublecomplex_t *a,
                           armpl_int_t lda, double vl, double vu,
                           armpl_int_t il, armpl_int_t iu, double abstol,
                           armpl_int_t *m, double *w,
                           armpl_doublecomplex_t *z, armpl_int_t ldz,
                           armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheevx* and *LAPACKE_zheevx*. It also exists with a native Fortran interface as *zheevx*.

4.19.2019 LAPACKE_zheevx_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zheevx_2stage(armpl_int_t matrix_layout, char jobz,
                                  char range, char uplo, armpl_int_t n,
                                  armpl_doublecomplex_t *a, armpl_int_t lda,
                                  double vl, double vu, armpl_int_t il,
                                  armpl_int_t iu, double abstol,
                                  armpl_int_t *m, double *w,
                                  armpl_doublecomplex_t *z, armpl_int_t ldz,
                                  armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheevx_2stage* and *LAPACKE_zheevx_2stage*. It also exists with a native Fortran interface as *zheevx_2stage*.

4.19.2020 LAPACKE_zheevx_2stage_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zheevx_2stage_work(armpl_int_t matrix_layout, char jobz,
                                       char range, char uplo, armpl_int_t n,
                                       armpl_doublecomplex_t *a,
                                       armpl_int_t lda, double vl, double vu,
                                       armpl_int_t il, armpl_int_t iu,
                                       double abstol, armpl_int_t *m,
                                       double *w, armpl_doublecomplex_t *z,
                                       armpl_int_t ldz,
                                       armpl_doublecomplex_t *work,
                                       armpl_int_t lwork, double *rwork,
                                       armpl_int_t *iwork,
                                       armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheevx_2stage_work* and *LAPACKE_zheevx_2stage_work*.

4.19.2021 LAPACKE_zheevx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zheevx_work(armpl_int_t matrix_layout, char jobz,
                                char range, char uplo, armpl_int_t n,
                                armpl_doublecomplex_t *a, armpl_int_t lda,
                                double vl, double vu, armpl_int_t il,
                                armpl_int_t iu, double abstol, armpl_int_t *m,
                                double *w, armpl_doublecomplex_t *z,
                                armpl_int_t ldz, armpl_doublecomplex_t *work,
                                armpl_int_t lwork, double *rwork,
                                armpl_int_t *iwork, armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheevx_work* and *LAPACKE_zheevx_work*.

4.19.2022 LAPACKE_zhegst

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhegst(armpl_int_t matrix_layout, armpl_int_t itype,
                           char uplo, armpl_int_t n, armpl_doublecomplex_t *a,
                           armpl_int_t lda, const armpl_doublecomplex_t *b,
                           armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chegst* and *LAPACKE_zhegst*. It also exists with a native Fortran interface as *zhegst*.

4.19.2023 LAPACKE_zhegst_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhegst_work(armpl_int_t matrix_layout, armpl_int_t itype,
                                char uplo, armpl_int_t n,
                                armpl_doublecomplex_t *a, armpl_int_t lda,
                                const armpl_doublecomplex_t *b,
                                armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chegst_work* and *LAPACKE_zhegst_work*.

4.19.2024 LAPACKE_zhegv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhegv(armpl_int_t matrix_layout, armpl_int_t itype,
                          char jobz, char uplo, armpl_int_t n,
                          armpl_doublecomplex_t *a, armpl_int_t lda,
                          armpl_doublecomplex_t *b, armpl_int_t ldb,
                          double *w);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chegv* and *LAPACKE_zhegv*. It also exists with a native Fortran interface as *zhegv*.

4.19.2025 LAPACKE_zhegv_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhegv_2stage(armpl_int_t matrix_layout, armpl_int_t itype,
                                 char jobz, char uplo, armpl_int_t n,
                                 armpl_doublecomplex_t *a, armpl_int_t lda,
                                 armpl_doublecomplex_t *b, armpl_int_t ldb,
                                 double *w);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chegv_2stage* and *LAPACKE_zhegv_2stage*. It also exists with a native Fortran interface as *zhegv_2stage*.

4.19.2026 LAPACKE_zhegv_2stage_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhegv_2stage_work(armpl_int_t matrix_layout,
                                      armpl_int_t itype, char jobz, char uplo,
                                      armpl_int_t n, armpl_doublecomplex_t *a,
                                      armpl_int_t lda,
                                      armpl_doublecomplex_t *b,
                                      armpl_int_t ldb, double *w,
                                      armpl_doublecomplex_t *work,
                                      armpl_int_t lwork, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chegv_2stage_work* and *LAPACKE_zhegv_2stage_work*.

4.19.2027 LAPACKE_zhegv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhegv_work(armpl_int_t matrix_layout, armpl_int_t itype,
                               char jobz, char uplo, armpl_int_t n,
                               armpl_doublecomplex_t *a, armpl_int_t lda,
                               armpl_doublecomplex_t *b, armpl_int_t ldb,
                               double *w, armpl_doublecomplex_t *work,
                               armpl_int_t lwork, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chegv_work* and *LAPACKE_zhegv_work*.

4.19.2028 LAPACKE_zhegvd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhegvd(armpl_int_t matrix_layout, armpl_int_t itype,
                           char jobz, char uplo, armpl_int_t n,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           armpl_doublecomplex_t *b, armpl_int_t ldb,
                           double *w);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chegvd* and *LAPACKE_zhegvd*. It also exists with a native Fortran interface as *zhegvd*.

4.19.2029 LAPACKE_zhegvd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhegvd_work(armpl_int_t matrix_layout, armpl_int_t itype,
                                char jobz, char uplo, armpl_int_t n,
                                armpl_doublecomplex_t *a, armpl_int_t lda,
                                armpl_doublecomplex_t *b, armpl_int_t ldb,
                                double *w, armpl_doublecomplex_t *work,
                                armpl_int_t lwork, double *rwork,
                                armpl_int_t lrwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chegvd_work* and *LAPACKE_zhegvd_work*.

4.19.2030 LAPACKE_zhegvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhegvx(armpl_int_t matrix_layout, armpl_int_t itype,
                           char jobz, char range, char uplo, armpl_int_t n,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           armpl_doublecomplex_t *b, armpl_int_t ldb,
                           double vl, double vu, armpl_int_t il,
                           armpl_int_t iu, double abstol, armpl_int_t *m,
                           double *w, armpl_doublecomplex_t *z,
                           armpl_int_t ldz, armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chegvx* and *LAPACKE_zhegvx*. It also exists with a native Fortran interface as *zhegvx*.

4.19.2031 LAPACKE_zhegvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhegvx_work(armpl_int_t matrix_layout, armpl_int_t itype,
                                char jobz, char range, char uplo,
                                armpl_int_t n, armpl_doublecomplex_t *a,
                                armpl_int_t lda, armpl_doublecomplex_t *b,
                                armpl_int_t ldb, double vl, double vu,
                                armpl_int_t il, armpl_int_t iu, double abstol,
                                armpl_int_t *m, double *w,
                                armpl_doublecomplex_t *z, armpl_int_t ldz,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork, double *rwork,
                                armpl_int_t *iwork, armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chegvx_work* and *LAPACKE_zhegvx_work*.

4.19.2032 LAPACKE_zherfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zherfs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_doublecomplex_t *a, armpl_int_t lda,
                           const armpl_doublecomplex_t *af, armpl_int_t ldaf,
                           const armpl_int_t *ipiv,
```

(continues on next page)

(continued from previous page)

```
                                const armpl_doublecomplex_t *b, armpl_int_t ldb,
                                armpl_doublecomplex_t *x, armpl_int_t ldx,
                                double *ferr, double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cherfs* and *LAPACKE_zherfs*. It also exists with a native Fortran interface as *zherfs*.

4.19.2033 LAPACKE_zherfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zherfs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const armpl_doublecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_doublecomplex_t *af,
                                armpl_int_t ldaf, const armpl_int_t *ipiv,
                                const armpl_doublecomplex_t *b,
                                armpl_int_t ldb, armpl_doublecomplex_t *x,
                                armpl_int_t ldx, double *ferr, double *berr,
                                armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cherfs_work* and *LAPACKE_zherfs_work*.

4.19.2034 LAPACKE_zherfsx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zherfsx(armpl_int_t matrix_layout, char uplo, char equed,
                            armpl_int_t n, armpl_int_t nrhs,
                            const armpl_doublecomplex_t *a, armpl_int_t lda,
                            const armpl_doublecomplex_t *af, armpl_int_t ldaf,
                            const armpl_int_t *ipiv, const double *s,
                            const armpl_doublecomplex_t *b, armpl_int_t ldb,
                            armpl_doublecomplex_t *x, armpl_int_t ldx,
                            double *rcond, double *berr,
                            armpl_int_t n_err_bnds, double *err_bnds_norm,
                            double *err_bnds_comp, armpl_int_t nparams,
                            double *params);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cherfsx* and *LAPACKE_zherfsx*. It also exists with a native Fortran interface as *zherfsx*.

4.19.2035 LAPACKE_zherfsx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zherfsx_work(armpl_int_t matrix_layout, char uplo,
                                 char equed, armpl_int_t n, armpl_int_t nrhs,
                                 const armpl_doublecomplex_t *a,
                                 armpl_int_t lda,
                                 const armpl_doublecomplex_t *af,
                                 armpl_int_t ldaf, const armpl_int_t *ipiv,
                                 const double *s,
                                 const armpl_doublecomplex_t *b,
                                 armpl_int_t ldb, armpl_doublecomplex_t *x,
                                 armpl_int_t ldx, double *rcond, double *berr,
                                 armpl_int_t n_err_bnds,
                                 double *err_bnds_norm, double *err_bnds_comp,
                                 armpl_int_t nparams, double *params,
                                 armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cherfsx_work* and *LAPACKE_zherfsx_work*.

4.19.2036 LAPACKE_zhesv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhesv(armpl_int_t matrix_layout, char uplo, armpl_int_t n,
                          armpl_int_t nrhs, armpl_doublecomplex_t *a,
                          armpl_int_t lda, armpl_int_t *ipiv,
                          armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chesv* and *LAPACKE_zhesv*. It also exists with a native Fortran interface as *zhesv*.

4.19.2037 LAPACKE_zhesv_aa

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhesv_aa(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, armpl_int_t nrhs,
                             armpl_doublecomplex_t *a, armpl_int_t lda,
                             armpl_int_t *ipiv, armpl_doublecomplex_t *b,
                             armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chesv_aa* and *LAPACKE_zhesv_aa*. It also exists with a native Fortran interface as *zhesv_aa*.

4.19.2038 LAPACKE_zhesv_aa_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhesv_aa_2stage(armpl_int_t matrix_layout, char uplo,
                                    armpl_int_t n, armpl_int_t nrhs,
                                    armpl_doublecomplex_t *a, armpl_int_t lda,
                                    armpl_doublecomplex_t *tb,
                                    armpl_int_t ltb, armpl_int_t *ipiv,
                                    armpl_int_t *ipiv2,
                                    armpl_doublecomplex_t *b,
                                    armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chesv_aa_2stage* and *LAPACKE_zhesv_aa_2stage*. It also exists with a native Fortran interface as *zhesv_aa_2stage*.

4.19.2039 LAPACKE_zhesv_aa_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhesv_aa_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, armpl_int_t nrhs,
                                  armpl_doublecomplex_t *a, armpl_int_t lda,
                                  armpl_int_t *ipiv, armpl_doublecomplex_t *b,
                                  armpl_int_t ldb,
                                  armpl_doublecomplex_t *work,
                                  armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chesv_aa_work* and *LAPACKE_zhesv_aa_work*.

4.19.2040 LAPACKE_zhesv_rk

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhesv_rk(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, armpl_int_t nrhs,
                             armpl_doublecomplex_t *a, armpl_int_t lda,
                             armpl_doublecomplex_t *e, armpl_int_t *ipiv,
                             armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chesv_rk* and *LAPACKE_zhesv_rk*. It also exists with a native Fortran interface as *zhesv_rk*.

4.19.2041 LAPACKE_zhesv_rk_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhesv_rk_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, armpl_int_t nrhs,
                                  armpl_doublecomplex_t *a, armpl_int_t lda,
                                  armpl_doublecomplex_t *e, armpl_int_t *ipiv,
                                  armpl_doublecomplex_t *b, armpl_int_t ldb,
                                  armpl_doublecomplex_t *work,
                                  armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chesv_rk_work* and *LAPACKE_zhesv_rk_work*.

4.19.2042 LAPACKE_zhesv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhesv_work(armpl_int_t matrix_layout, char uplo,
                               armpl_int_t n, armpl_int_t nrhs,
                               armpl_doublecomplex_t *a, armpl_int_t lda,
                               armpl_int_t *ipiv, armpl_doublecomplex_t *b,
                               armpl_int_t ldb, armpl_doublecomplex_t *work,
                               armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chesv_work* and *LAPACKE_zhesv_work*.

4.19.2043 LAPACKE_zhesvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhesvx(armpl_int_t matrix_layout, char fact, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_doublecomplex_t *a, armpl_int_t lda,
                           armpl_doublecomplex_t *af, armpl_int_t ldaf,
                           armpl_int_t *ipiv, const armpl_doublecomplex_t *b,
                           armpl_int_t ldb, armpl_doublecomplex_t *x,
                           armpl_int_t ldx, double *rcond, double *ferr,
                           double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chesvx* and *LAPACKE_zhesvx*. It also exists with a native Fortran interface as *zhesvx*.

4.19.2044 LAPACKE_zhesvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhesvx_work(armpl_int_t matrix_layout, char fact,
                                char uplo, armpl_int_t n, armpl_int_t nrhs,
                                const armpl_doublecomplex_t *a,
                                armpl_int_t lda, armpl_doublecomplex_t *af,
                                armpl_int_t ldaf, armpl_int_t *ipiv,
                                const armpl_doublecomplex_t *b,
                                armpl_int_t ldb, armpl_doublecomplex_t *x,
                                armpl_int_t ldx, double *rcond, double *ferr,
                                double *berr, armpl_doublecomplex_t *work,
                                armpl_int_t lwork, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chesvx_work* and *LAPACKE_zhesvx_work*.

4.19.2045 LAPACKE_zhesvxx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhesvxx(armpl_int_t matrix_layout, char fact, char uplo,
                            armpl_int_t n, armpl_int_t nrhs,
                            armpl_doublecomplex_t *a, armpl_int_t lda,
                            armpl_doublecomplex_t *af, armpl_int_t ldaf,
                            armpl_int_t *ipiv, char *equed, double *s,
                            armpl_doublecomplex_t *b, armpl_int_t ldb,
                            armpl_doublecomplex_t *x, armpl_int_t ldx,
                            double *rcond, double *rpvgrw, double *berr,
                            armpl_int_t n_err_bnds, double *err_bnds_norm,
                            double *err_bnds_comp, armpl_int_t nparams,
                            double *params);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chesvxx* and *LAPACKE_zhesvxx*. It also exists with a native Fortran interface as *zhesvxx*.

4.19.2046 LAPACKE_zhesvxx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhesvxx_work(armpl_int_t matrix_layout, char fact,
                                 char uplo, armpl_int_t n, armpl_int_t nrhs,
                                 armpl_doublecomplex_t *a, armpl_int_t lda,
                                 armpl_doublecomplex_t *af, armpl_int_t ldaf,
                                 armpl_int_t *ipiv, char *equed, double *s,
                                 armpl_doublecomplex_t *b, armpl_int_t ldb,
                                 armpl_doublecomplex_t *x, armpl_int_t ldx,
                                 double *rcond, double *rpvgrw, double *berr,
                                 armpl_int_t n_err_bnds,
                                 double *err_bnds_norm, double *err_bnds_comp,
                                 armpl_int_t nparams, double *params,
                                 armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chesvxx_work* and *LAPACKE_zhesvxx_work*.

4.19.2047 LAPACKE_zheswapr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zheswapr(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, armpl_doublecomplex_t *a,
                             armpl_int_t lda, armpl_int_t i1,
                             armpl_int_t i2);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheswapr* and *LAPACKE_zheswapr*. It also exists with a native Fortran interface as *zheswapr*.

4.19.2048 LAPACKE_zheswapr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zheswapr_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, armpl_doublecomplex_t *a,
                                  armpl_int_t lda, armpl_int_t i1,
                                  armpl_int_t i2);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cheswapr_work* and *LAPACKE_zheswapr_work*.

4.19.2049 LAPACKE_zhetrd

4.19.2050 LAPACKE_zhetrd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhetrd_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_doublecomplex_t *a,
                                armpl_int_t lda, double *d, double *e,
                                armpl_doublecomplex_t *tau,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrd_work* and *LAPACKE_zhetrd_work*.

4.19.2051 LAPACKE_zhetrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhetrf(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_doublecomplex_t *a,
                           armpl_int_t lda, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrf* and *LAPACKE_zhetrf*. It also exists with a native Fortran interface as *zhetrf*.

4.19.2052 LAPACKE_zhetrf_aa

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhetrf_aa(armpl_int_t matrix_layout, char uplo,
                              armpl_int_t n, armpl_doublecomplex_t *a,
                              armpl_int_t lda, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrf_aa* and *LAPACKE_zhetrf_aa*. It also exists with a native Fortran interface as *zhetrf_aa*.

4.19.2053 LAPACKE_zhetrf_aa_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhetrf_aa_2stage(armpl_int_t matrix_layout, char uplo,
                                     armpl_int_t n, armpl_doublecomplex_t *a,
                                     armpl_int_t lda,
                                     armpl_doublecomplex_t *tb,
                                     armpl_int_t ltb, armpl_int_t *ipiv,
                                     armpl_int_t *ipiv2);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrf_aa_2stage* and *LAPACKE_zhetrf_aa_2stage*. It also exists with a native Fortran interface as *zhetrf_aa_2stage*.

4.19.2054 LAPACKE_zhetrf_aa_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhetrf_aa_work(armpl_int_t matrix_layout, char uplo,
                                   armpl_int_t n, armpl_doublecomplex_t *a,
                                   armpl_int_t lda, armpl_int_t *ipiv,
                                   armpl_doublecomplex_t *work,
                                   armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrf_aa_work* and *LAPACKE_zhetrf_aa_work*.

4.19.2055 LAPACKE_zhetrf_rk

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhetrf_rk(armpl_int_t matrix_layout, char uplo,
                              armpl_int_t n, armpl_doublecomplex_t *a,
                              armpl_int_t lda, armpl_doublecomplex_t *e,
                              armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrf_rk* and *LAPACKE_zhetrf_rk*. It also exists with a native Fortran interface as *zhetrf_rk*.

4.19.2056 LAPACKE_zhetrf_rk_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhetrf_rk_work(armpl_int_t matrix_layout, char uplo,
                                   armpl_int_t n, armpl_doublecomplex_t *a,
                                   armpl_int_t lda, armpl_doublecomplex_t *e,
                                   armpl_int_t *ipiv,
                                   armpl_doublecomplex_t *work,
                                   armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrf_rk_work* and *LAPACKE_zhetrf_rk_work*.

4.19.2057 LAPACKE_zhetrf_rook

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhetrf_rook(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_doublecomplex_t *a,
                                armpl_int_t lda, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrf_rook* and *LAPACKE_zhetrf_rook*. It also exists with a native Fortran interface as *zhetrf_rook*.

4.19.2058 LAPACKE_zhetrf_rook_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhetrf_rook_work(armpl_int_t matrix_layout, char uplo,
                                     armpl_int_t n, armpl_doublecomplex_t *a,
                                     armpl_int_t lda, armpl_int_t *ipiv,
                                     armpl_doublecomplex_t *work,
                                     armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrf_rook_work* and *LAPACKE_zhetrf_rook_work*.

4.19.2059 LAPACKE_zhetrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhetrf_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_doublecomplex_t *a,
                                armpl_int_t lda, armpl_int_t *ipiv,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrf_work* and *LAPACKE_zhetrf_work*.

4.19.2060 LAPACKE_zhetri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhetri(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_doublecomplex_t *a,
                           armpl_int_t lda, const armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetri* and *LAPACKE_zhetri*. It also exists with a native Fortran interface as *zhetri*.

4.19.2061 LAPACKE_zhetri2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhetri2(armpl_int_t matrix_layout, char uplo,
                            armpl_int_t n, armpl_doublecomplex_t *a,
                            armpl_int_t lda, const armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetri2* and *LAPACKE_zhetri2*. It also exists with a native Fortran interface as *zhetri2*.

4.19.2062 LAPACKE_zhetri2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhetri2_work(armpl_int_t matrix_layout, char uplo,
                                 armpl_int_t n, armpl_doublecomplex_t *a,
                                 armpl_int_t lda, const armpl_int_t *ipiv,
                                 armpl_doublecomplex_t *work,
                                 armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetri2_work* and *LAPACKE_zhetri2_work*.

4.19.2063 LAPACKE_zhetri2x

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhetri2x(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, armpl_doublecomplex_t *a,
                             armpl_int_t lda, const armpl_int_t *ipiv,
                             armpl_int_t nb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetri2x* and *LAPACKE_zhetri2x*. It also exists with a native Fortran interface as *zhetri2x*.

4.19.2064 LAPACKE_zhetri2x_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhetri2x_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, armpl_doublecomplex_t *a,
                                  armpl_int_t lda, const armpl_int_t *ipiv,
                                  armpl_doublecomplex_t *work,
                                  armpl_int_t nb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetri2x_work* and *LAPACKE_zhetri2x_work*.

4.19.2065 LAPACKE_zhetri_3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhetri_3(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, armpl_doublecomplex_t *a,
                             armpl_int_t lda, const armpl_doublecomplex_t *e,
                             const armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetri_3* and *LAPACKE_zhetri_3*. It also exists with a native Fortran interface as *zhetri_3*.

4.19.2066 LAPACKE_zhetri_3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhetri_3_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, armpl_doublecomplex_t *a,
                                  armpl_int_t lda,
                                  const armpl_doublecomplex_t *e,
                                  const armpl_int_t *ipiv,
                                  armpl_doublecomplex_t *work,
                                  armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetri_3_work* and *LAPACKE_zhetri_3_work*.

4.19.2067 LAPACKE_zhetri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhetri_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_doublecomplex_t *a,
                                armpl_int_t lda, const armpl_int_t *ipiv,
                                armpl_doublecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetri_work* and *LAPACKE_zhetri_work*.

4.19.2068 LAPACKE_zhetrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhetrs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_doublecomplex_t *a, armpl_int_t lda,
                           const armpl_int_t *ipiv, armpl_doublecomplex_t *b,
                           armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrs* and *LAPACKE_zhetrs*. It also exists with a native Fortran interface as *zhetrs*.

4.19.2069 LAPACKE_zhetrs2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhetrs2(armpl_int_t matrix_layout, char uplo,
                            armpl_int_t n, armpl_int_t nrhs,
                            const armpl_doublecomplex_t *a, armpl_int_t lda,
                            const armpl_int_t *ipiv, armpl_doublecomplex_t *b,
                            armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrs2* and *LAPACKE_zhetrs2*. It also exists with a native Fortran interface as *zhetrs2*.

4.19.2070 LAPACKE_zhetrs2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhetrs2_work(armpl_int_t matrix_layout, char uplo,
                                 armpl_int_t n, armpl_int_t nrhs,
                                 const armpl_doublecomplex_t *a,
                                 armpl_int_t lda, const armpl_int_t *ipiv,
                                 armpl_doublecomplex_t *b, armpl_int_t ldb,
                                 armpl_doublecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrs2_work* and *LAPACKE_zhetrs2_work*.

4.19.2071 LAPACKE_zhetrs_3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhetrs_3(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, armpl_int_t nrhs,
                             const armpl_doublecomplex_t *a, armpl_int_t lda,
                             const armpl_doublecomplex_t *e,
                             const armpl_int_t *ipiv,
                             armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrs_3* and *LAPACKE_zhetrs_3*. It also exists with a native Fortran interface as *zhetrs_3*.

4.19.2072 LAPACKE_zhetrs_3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhetrs_3_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, armpl_int_t nrhs,
                                  const armpl_doublecomplex_t *a,
                                  armpl_int_t lda,
```

(continues on next page)

(continued from previous page)

```
                                    const armpl_doublecomplex_t *e,
                                    const armpl_int_t *ipiv,
                                    armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrs_3_work* and *LAPACKE_zhetrs_3_work*.

4.19.2073 LAPACKE_zhetrs_aa

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhetrs_aa(armpl_int_t matrix_layout, char uplo,
                              armpl_int_t n, armpl_int_t nrhs,
                              const armpl_doublecomplex_t *a, armpl_int_t lda,
                              const armpl_int_t *ipiv,
                              armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrs_aa* and *LAPACKE_zhetrs_aa*. It also exists with a native Fortran interface as *zhetrs_aa*.

4.19.2074 LAPACKE_zhetrs_aa_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhetrs_aa_2stage(armpl_int_t matrix_layout, char uplo,
                                     armpl_int_t n, armpl_int_t nrhs,
                                     armpl_doublecomplex_t *a,
                                     armpl_int_t lda,
                                     armpl_doublecomplex_t *tb,
                                     armpl_int_t ltb, armpl_int_t *ipiv,
                                     armpl_int_t *ipiv2,
                                     armpl_doublecomplex_t *b,
                                     armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrs_aa_2stage* and *LAPACKE_zhetrs_aa_2stage*. It also exists with a native Fortran interface as *zhetrs_aa_2stage*.

4.19.2075 LAPACKE_zhetrs_aa_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhetrs_aa_work(armpl_int_t matrix_layout, char uplo,
                                   armpl_int_t n, armpl_int_t nrhs,
                                   const armpl_doublecomplex_t *a,
                                   armpl_int_t lda, const armpl_int_t *ipiv,
                                   armpl_doublecomplex_t *b, armpl_int_t ldb,
                                   armpl_doublecomplex_t *work,
                                   armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrs_aa_work* and *LAPACKE_zhetrs_aa_work*.

4.19.2076 LAPACKE_zhetrs_rook

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhetrs_rook(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const armpl_doublecomplex_t *a,
                                armpl_int_t lda, const armpl_int_t *ipiv,
                                armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrs_rook* and *LAPACKE_zhetrs_rook*. It also exists with a native Fortran interface as *zhetrs_rook*.

4.19.2077 LAPACKE_zhetrs_rook_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhetrs_rook_work(armpl_int_t matrix_layout, char uplo,
                                     armpl_int_t n, armpl_int_t nrhs,
                                     const armpl_doublecomplex_t *a,
                                     armpl_int_t lda, const armpl_int_t *ipiv,
                                     armpl_doublecomplex_t *b,
                                     armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrs_rook_work* and *LAPACKE_zhetrs_rook_work*.

4.19.2078 LAPACKE_zhetrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhetrs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const armpl_doublecomplex_t *a,
                                armpl_int_t lda, const armpl_int_t *ipiv,
                                armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chetrs_work* and *LAPACKE_zhetrs_work*.

4.19.2079 LAPACKE_zhfrk

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhfrk(armpl_int_t matrix_layout, char transr, char uplo,
                          char trans, armpl_int_t n, armpl_int_t k,
                          double alpha, const armpl_doublecomplex_t *a,
                          armpl_int_t lda, double beta,
                          armpl_doublecomplex_t *c);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chfrk* and *LAPACKE_zhfrk*. It also exists with a native Fortran interface as *zhfrk*.

4.19.2080 LAPACKE_zhfrk_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhfrk_work(armpl_int_t matrix_layout, char transr,
                               char uplo, char trans, armpl_int_t n,
                               armpl_int_t k, double alpha,
                               const armpl_doublecomplex_t *a,
                               armpl_int_t lda, double beta,
                               armpl_doublecomplex_t *c);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chfrk_work* and *LAPACKE_zhfrk_work*.

4.19.2081 LAPACKE_zhgeqz

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhgeqz(armpl_int_t matrix_layout, char job, char compq,
                           char compz, armpl_int_t n, armpl_int_t ilo,
                           armpl_int_t ihi, armpl_doublecomplex_t *h,
                           armpl_int_t ldh, armpl_doublecomplex_t *t,
                           armpl_int_t ldt, armpl_doublecomplex_t *alpha,
                           armpl_doublecomplex_t *beta,
                           armpl_doublecomplex_t *q, armpl_int_t ldq,
                           armpl_doublecomplex_t *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chgeqz*, *LAPACKE_dhgeqz*, *LAPACKE_shgeqz* and *LAPACKE_zhgeqz*. It also exists with a native Fortran interface as *zhgeqz*.

4.19.2082 LAPACKE_zhgeqz_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhgeqz_work(armpl_int_t matrix_layout, char job,
                                char compq, char compz, armpl_int_t n,
                                armpl_int_t ilo, armpl_int_t ihi,
                                armpl_doublecomplex_t *h, armpl_int_t ldh,
                                armpl_doublecomplex_t *t, armpl_int_t ldt,
                                armpl_doublecomplex_t *alpha,
                                armpl_doublecomplex_t *beta,
                                armpl_doublecomplex_t *q, armpl_int_t ldq,
                                armpl_doublecomplex_t *z, armpl_int_t ldz,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chgeqz_work*, *LAPACKE_dhgeqz_work*, *LAPACKE_shgeqz_work* and *LAPACKE_zhgeqz_work*.

4.19.2083 LAPACKE_zhpcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhpcon(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, const armpl_doublecomplex_t *ap,
                           const armpl_int_t *ipiv, double anorm,
                           double *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpcon* and *LAPACKE_zhpcon*. It also exists with a native Fortran interface as *zhpcon*.

4.19.2084 LAPACKE_zhpcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhpcon_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n,
                                const armpl_doublecomplex_t *ap,
                                const armpl_int_t *ipiv, double anorm,
                                double *rcond, armpl_doublecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpcon_work* and *LAPACKE_zhpcon_work*.

4.19.2085 LAPACKE_zhpev

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhpev(armpl_int_t matrix_layout, char jobz, char uplo,
                          armpl_int_t n, armpl_doublecomplex_t *ap, double *w,
                          armpl_doublecomplex_t *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpev* and *LAPACKE_zhpev*. It also exists with a native Fortran interface as *zhpev*.

4.19.2086 LAPACKE_zhpev_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhpev_work(armpl_int_t matrix_layout, char jobz,
                               char uplo, armpl_int_t n,
                               armpl_doublecomplex_t *ap, double *w,
                               armpl_doublecomplex_t *z, armpl_int_t ldz,
                               armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpev_work* and *LAPACKE_zhpev_work*.

4.19.2087 LAPACKE_zhpevd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhpevd(armpl_int_t matrix_layout, char jobz, char uplo,
                           armpl_int_t n, armpl_doublecomplex_t *ap,
                           double *w, armpl_doublecomplex_t *z,
                           armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpevd* and *LAPACKE_zhpevd*. It also exists with a native Fortran interface as *zhpevd*.

4.19.2088 LAPACKE_zhpevd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhpevd_work(armpl_int_t matrix_layout, char jobz,
                                char uplo, armpl_int_t n,
                                armpl_doublecomplex_t *ap, double *w,
                                armpl_doublecomplex_t *z, armpl_int_t ldz,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork, double *rwork,
                                armpl_int_t lrwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpevd_work* and *LAPACKE_zhpevd_work*.

4.19.2089 LAPACKE_zhpevx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhpevx(armpl_int_t matrix_layout, char jobz, char range,
                           char uplo, armpl_int_t n,
```

(continues on next page)

(continued from previous page)

```
                                armpl_doublecomplex_t *ap, double vl, double vu,
                                armpl_int_t il, armpl_int_t iu, double abstol,
                                armpl_int_t *m, double *w,
                                armpl_doublecomplex_t *z, armpl_int_t ldz,
                                armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpevx* and *LAPACKE_zhpevx*. It also exists with a native Fortran interface as *zhpevx*.

4.19.2090 LAPACKE_zhpevx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhpevx_work(armpl_int_t matrix_layout, char jobz,
                                char range, char uplo, armpl_int_t n,
                                armpl_doublecomplex_t *ap, double vl,
                                double vu, armpl_int_t il, armpl_int_t iu,
                                double abstol, armpl_int_t *m, double *w,
                                armpl_doublecomplex_t *z, armpl_int_t ldz,
                                armpl_doublecomplex_t *work, double *rwork,
                                armpl_int_t *iwork, armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpevx_work* and *LAPACKE_zhpevx_work*.

4.19.2091 LAPACKE_zhpgst

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhpgst(armpl_int_t matrix_layout, armpl_int_t itype,
                           char uplo, armpl_int_t n,
                           armpl_doublecomplex_t *ap,
                           const armpl_doublecomplex_t *bp);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpgst* and *LAPACKE_zhpgst*. It also exists with a native Fortran interface as *zhpgst*.

4.19.2092 LAPACKE_zhpgst_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhpgst_work(armpl_int_t matrix_layout, armpl_int_t itype,
                                char uplo, armpl_int_t n,
                                armpl_doublecomplex_t *ap,
                                const armpl_doublecomplex_t *bp);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpgst_work* and *LAPACKE_zhpgst_work*.

4.19.2093 LAPACKE_zhpgv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhpgv(armpl_int_t matrix_layout, armpl_int_t itype,
                          char jobz, char uplo, armpl_int_t n,
                          armpl_doublecomplex_t *ap,
                          armpl_doublecomplex_t *bp, double *w,
                          armpl_doublecomplex_t *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpgv* and *LAPACKE_zhpgv*. It also exists with a native Fortran interface as *zhpgv*.

4.19.2094 LAPACKE_zhpgv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhpgv_work(armpl_int_t matrix_layout, armpl_int_t itype,
                               char jobz, char uplo, armpl_int_t n,
                               armpl_doublecomplex_t *ap,
                               armpl_doublecomplex_t *bp, double *w,
                               armpl_doublecomplex_t *z, armpl_int_t ldz,
                               armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpgv_work* and *LAPACKE_zhpgv_work*.

4.19.2095 LAPACKE_zhpgvd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhpgvd(armpl_int_t matrix_layout, armpl_int_t itype,
                           char jobz, char uplo, armpl_int_t n,
                           armpl_doublecomplex_t *ap,
                           armpl_doublecomplex_t *bp, double *w,
                           armpl_doublecomplex_t *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpgvd* and *LAPACKE_zhpgvd*. It also exists with a native Fortran interface as *zhpgvd*.

4.19.2096 LAPACKE_zhpgvd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhpgvd_work(armpl_int_t matrix_layout, armpl_int_t itype,
                                char jobz, char uplo, armpl_int_t n,
                                armpl_doublecomplex_t *ap,
                                armpl_doublecomplex_t *bp, double *w,
                                armpl_doublecomplex_t *z, armpl_int_t ldz,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork, double *rwork,
                                armpl_int_t lrwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpgvd_work* and *LAPACKE_zhpgvd_work*.

4.19.2097 LAPACKE_zhpgvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhpgvx(armpl_int_t matrix_layout, armpl_int_t itype,
                           char jobz, char range, char uplo, armpl_int_t n,
                           armpl_doublecomplex_t *ap,
                           armpl_doublecomplex_t *bp, double vl, double vu,
                           armpl_int_t il, armpl_int_t iu, double abstol,
                           armpl_int_t *m, double *w,
                           armpl_doublecomplex_t *z, armpl_int_t ldz,
                           armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpgvx* and *LAPACKE_zhpgvx*. It also exists with a native Fortran interface as *zhpgvx*.

4.19.2098 LAPACKE_zhpgvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhpgvx_work(armpl_int_t matrix_layout, armpl_int_t itype,
                                char jobz, char range, char uplo,
                                armpl_int_t n, armpl_doublecomplex_t *ap,
                                armpl_doublecomplex_t *bp, double vl,
                                double vu, armpl_int_t il, armpl_int_t iu,
                                double abstol, armpl_int_t *m, double *w,
                                armpl_doublecomplex_t *z, armpl_int_t ldz,
                                armpl_doublecomplex_t *work, double *rwork,
                                armpl_int_t *iwork, armpl_int_t *ifail);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpgvx_work* and *LAPACKE_zhpgvx_work*.

4.19.2099 LAPACKE_zhprfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhprfs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_doublecomplex_t *ap,
                           const armpl_doublecomplex_t *afp,
                           const armpl_int_t *ipiv,
                           const armpl_doublecomplex_t *b, armpl_int_t ldb,
                           armpl_doublecomplex_t *x, armpl_int_t ldx,
                           double *ferr, double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chprfs* and *LAPACKE_zhprfs*. It also exists with a native Fortran interface as *zhprfs*.

4.19.2100 LAPACKE_zhprfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhprfs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const armpl_doublecomplex_t *ap,
                                const armpl_doublecomplex_t *afp,
                                const armpl_int_t *ipiv,
                                const armpl_doublecomplex_t *b,
```

(continues on next page)

(continued from previous page)

```
                                armpl_int_t ldb, armpl_doublecomplex_t *x,
                                armpl_int_t ldx, double *ferr, double *berr,
                                armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chprfs_work* and *LAPACKE_zhprfs_work*.

4.19.2101 LAPACKE_zhpsv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhpsv(armpl_int_t matrix_layout, char uplo, armpl_int_t n,
                          armpl_int_t nrhs, armpl_doublecomplex_t *ap,
                          armpl_int_t *ipiv, armpl_doublecomplex_t *b,
                          armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpsv* and *LAPACKE_zhpsv*. It also exists with a native Fortran interface as *zhpsv*.

4.19.2102 LAPACKE_zhpsv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhpsv_work(armpl_int_t matrix_layout, char uplo,
                               armpl_int_t n, armpl_int_t nrhs,
                               armpl_doublecomplex_t *ap, armpl_int_t *ipiv,
                               armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpsv_work* and *LAPACKE_zhpsv_work*.

4.19.2103 LAPACKE_zhpsvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhpsvx(armpl_int_t matrix_layout, char fact, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_doublecomplex_t *ap,
                           armpl_doublecomplex_t *afp, armpl_int_t *ipiv,
                           const armpl_doublecomplex_t *b, armpl_int_t ldb,
                           armpl_doublecomplex_t *x, armpl_int_t ldx,
                           double *rcond, double *ferr, double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpsvx* and *LAPACKE_zhpsvx*. It also exists with a native Fortran interface as *zhpsvx*.

4.19.2104 LAPACKE_zhpsvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhpsvx_work(armpl_int_t matrix_layout, char fact,
                                char uplo, armpl_int_t n, armpl_int_t nrhs,
                                const armpl_doublecomplex_t *ap,
                                armpl_doublecomplex_t *afp, armpl_int_t *ipiv,
                                const armpl_doublecomplex_t *b,
                                armpl_int_t ldb, armpl_doublecomplex_t *x,
                                armpl_int_t ldx, double *rcond, double *ferr,
                                double *berr, armpl_doublecomplex_t *work,
                                double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chpsvx_work* and *LAPACKE_zhpsvx_work*.

4.19.2105 LAPACKE_zhptrd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhptrd(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_doublecomplex_t *ap,
                           double *d, double *e, armpl_doublecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chptrd* and *LAPACKE_zhptrd*. It also exists with a native Fortran interface as *zhptrd*.

4.19.2106 LAPACKE_zhptrd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhptrd_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_doublecomplex_t *ap,
                                double *d, double *e,
                                armpl_doublecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chptrd_work* and *LAPACKE_zhptrd_work*.

4.19.2107 LAPACKE_zhptrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhptrf(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_doublecomplex_t *ap,
                           armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chptrf* and *LAPACKE_zhptrf*. It also exists with a native Fortran interface as *zhptrf*.

4.19.2108 LAPACKE_zhptrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhptrf_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_doublecomplex_t *ap,
                                armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chptrf_work* and *LAPACKE_zhptrf_work*.

4.19.2109 LAPACKE_zhptri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhptri(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_doublecomplex_t *ap,
                           const armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chptri* and *LAPACKE_zhptri*. It also exists with a native Fortran interface as *zhptri*.

4.19.2110 LAPACKE_zhptri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhptri_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_doublecomplex_t *ap,
                                const armpl_int_t *ipiv,
                                armpl_doublecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chptri_work* and *LAPACKE_zhptri_work*.

4.19.2111 LAPACKE_zhptrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhptrs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_doublecomplex_t *ap,
                           const armpl_int_t *ipiv, armpl_doublecomplex_t *b,
                           armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.2112 LAPACKE_zhptrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhptrs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const armpl_doublecomplex_t *ap,
                                const armpl_int_t *ipiv,
                                armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chptrs_work* and *LAPACKE_zhptrs_work*.

4.19.2113 LAPACKE_zhsein

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhsein(armpl_int_t matrix_layout, char job, char eigsrc,
                           char initv, const armpl_int_t *select,
                           armpl_int_t n, const armpl_doublecomplex_t *h,
                           armpl_int_t ldh, armpl_doublecomplex_t *w,
                           armpl_doublecomplex_t *vl, armpl_int_t ldvl,
                           armpl_doublecomplex_t *vr, armpl_int_t ldvr,
                           armpl_int_t mm, armpl_int_t *m,
                           armpl_int_t *ifaill, armpl_int_t *ifailr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chsein*, *LAPACKE_dhsein*, *LAPACKE_shsein* and *LAPACKE_zhsein*. It also exists with a native Fortran interface as *zhsein*.

4.19.2114 LAPACKE_zhsein_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhsein_work(armpl_int_t matrix_layout, char job,
                                char eigsrc, char initv,
                                const armpl_int_t *select, armpl_int_t n,
                                const armpl_doublecomplex_t *h,
                                armpl_int_t ldh, armpl_doublecomplex_t *w,
                                armpl_doublecomplex_t *vl, armpl_int_t ldvl,
                                armpl_doublecomplex_t *vr, armpl_int_t ldvr,
                                armpl_int_t mm, armpl_int_t *m,
                                armpl_doublecomplex_t *work, double *rwork,
                                armpl_int_t *ifaill, armpl_int_t *ifailr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chsein_work*, *LAPACKE_dhsein_work*, *LAPACKE_shsein_work* and *LAPACKE_zhsein_work*.

4.19.2115 LAPACKE_zhseqr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhseqr(armpl_int_t matrix_layout, char job, char compz,
                           armpl_int_t n, armpl_int_t ilo, armpl_int_t ihi,
                           armpl_doublecomplex_t *h, armpl_int_t ldh,
                           armpl_doublecomplex_t *w, armpl_doublecomplex_t *z,
                           armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chseqr*, *LAPACKE_dhseqr*, *LAPACKE_shseqr* and *LAPACKE_zhseqr*. It also exists with a native Fortran interface as *zhseqr*.

4.19.2116 LAPACKE_zhseqr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zhseqr_work(armpl_int_t matrix_layout, char job,
                                char compz, armpl_int_t n, armpl_int_t ilo,
                                armpl_int_t ihi, armpl_doublecomplex_t *h,
                                armpl_int_t ldh, armpl_doublecomplex_t *w,
                                armpl_doublecomplex_t *z, armpl_int_t ldz,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_chseqr_work*, *LAPACKE_dhseqr_work*, *LAPACKE_shseqr_work* and *LAPACKE_zhseqr_work*.

4.19.2117 LAPACKE_zlacgv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlacgv(armpl_int_t n, armpl_doublecomplex_t *x,
                           armpl_int_t incx);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clacgv* and *LAPACKE_zlacgv*. It also exists with a native Fortran interface as *zlacgv*.

4.19.2118 LAPACKE_zlacgv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlacgv_work(armpl_int_t n, armpl_doublecomplex_t *x,
                                armpl_int_t incx);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clacgv_work* and *LAPACKE_zlacgv_work*.

4.19.2119 LAPACKE_zlacn2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlacn2(armpl_int_t n, armpl_doublecomplex_t *v,
                           armpl_doublecomplex_t *x, double *est,
                           armpl_int_t *kase, armpl_int_t *isave);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clacn2*, *LAPACKE_dlacn2*, *LAPACKE_slacn2* and *LAPACKE_zlacn2*. It also exists with a native Fortran interface as *zlacn2*.

4.19.2120 LAPACKE_zlacn2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlacn2_work(armpl_int_t n, armpl_doublecomplex_t *v,
                                armpl_doublecomplex_t *x, double *est,
                                armpl_int_t *kase, armpl_int_t *isave);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clacn2_work*, *LAPACKE_dlacn2_work*, *LAPACKE_slacn2_work* and *LAPACKE_zlacn2_work*.

4.19.2121 LAPACKE_zlacp2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlacp2(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t m, armpl_int_t n, const double *a,
                           armpl_int_t lda, armpl_doublecomplex_t *b,
                           armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clacp2* and *LAPACKE_zlacp2*. It also exists with a native Fortran interface as *zlacp2*.

4.19.2122 LAPACKE_zlacp2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlacp2_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t m, armpl_int_t n, const double *a,
                                armpl_int_t lda, armpl_doublecomplex_t *b,
                                armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clacp2_work* and *LAPACKE_zlacp2_work*.

4.19.2123 LAPACKE_zlacpy

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlacpy(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t m, armpl_int_t n,
                           const armpl_doublecomplex_t *a, armpl_int_t lda,
                           armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clacpy*, *LAPACKE_dlacpy*, *LAPACKE_slacpy* and *LAPACKE_zlacpy*. It also exists with a native Fortran interface as *zlacpy*.

4.19.2124 LAPACKE_zlacpy_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlacpy_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t m, armpl_int_t n,
                                const armpl_doublecomplex_t *a,
                                armpl_int_t lda, armpl_doublecomplex_t *b,
                                armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clacpy_work*, *LAPACKE_dlacpy_work*, *LAPACKE_slacpy_work* and *LAPACKE_zlacpy_work*.

4.19.2125 LAPACKE_zlacrm

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlacrm(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, const armpl_doublecomplex_t *a,
                           armpl_int_t lda, const double *b, armpl_int_t ldb,
                           armpl_doublecomplex_t *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clacrm* and *LAPACKE_zlacrm*. It also exists with a native Fortran interface as *zlacrm*.

4.19.2126 LAPACKE_zlacrm_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlacrm_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, const armpl_doublecomplex_t *a,
                                armpl_int_t lda, const double *b,
                                armpl_int_t ldb, armpl_doublecomplex_t *c,
                                armpl_int_t ldc, double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clacrm_work* and *LAPACKE_zlacrm_work*.

4.19.2127 LAPACKE_zlag2c

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlag2c(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, const armpl_doublecomplex_t *a,
                           armpl_int_t lda, armpl_singlecomplex_t *sa,
                           armpl_int_t ldsa);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_zlag2c*. It also exists with a native Fortran interface as *zlag2c*.

4.19.2128 LAPACKE_zlag2c_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlag2c_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, const armpl_doublecomplex_t *a,
                                armpl_int_t lda, armpl_singlecomplex_t *sa,
                                armpl_int_t ldsa);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.2129 LAPACKE_zlagge

Syntax

Routine is deprecated. Please choose a suitable alternative.

4.19.2130 LAPACKE_zlagge_work

Syntax

Routine is deprecated. Please choose a suitable alternative.

4.19.2131 LAPACKE_zlaghe

Syntax

Routine is deprecated. Please choose a suitable alternative.

4.19.2132 LAPACKE_zlaghe_work

Syntax

Routine is deprecated. Please choose a suitable alternative.

4.19.2133 LAPACKE_zlagsy

Syntax

Routine is deprecated. Please choose a suitable alternative.

4.19.2134 LAPACKE_zlagsy_work

Syntax

Routine is deprecated. Please choose a suitable alternative.

4.19.2135 LAPACKE_zlange

Syntax

```
#include "armpl.h"

double LAPACKE_zlange(armpl_int_t matrix_layout, char norm, armpl_int_t m,
                      armpl_int_t n, const armpl_doublecomplex_t *a,
                      armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clange*, *LAPACKE_dlange*, *LAPACKE_slange* and *LAPACKE_zlange*. It also exists with a native Fortran interface as *zlange*.

4.19.2136 LAPACKE_zlange_work

Syntax

```
#include "armpl.h"

double LAPACKE_zlange_work(armpl_int_t matrix_layout, char norm,
                           armpl_int_t m, armpl_int_t n,
                           const armpl_doublecomplex_t *a, armpl_int_t lda,
                           double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clange_work*, *LAPACKE_dlange_work*, *LAPACKE_slange_work* and *LAPACKE_zlange_work*.

4.19.2137 LAPACKE_zlanhe

Syntax

```
#include "armpl.h"

double LAPACKE_zlanhe(armpl_int_t matrix_layout, char norm, char uplo,
                      armpl_int_t n, const armpl_doublecomplex_t *a,
                      armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clanhe* and *LAPACKE_zlanhe*. It also exists with a native Fortran interface as *zlanhe*.

4.19.2138 LAPACKE_zlanhe_work

Syntax

```
#include "armpl.h"

double LAPACKE_zlanhe_work(armpl_int_t matrix_layout, char norm, char uplo,
                           armpl_int_t n, const armpl_doublecomplex_t *a,
                           armpl_int_t lda, double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clanhe_work* and *LAPACKE_zlanhe_work*.

4.19.2139 LAPACKE_zlansy

Syntax

```
#include "armpl.h"

double LAPACKE_zlansy(armpl_int_t matrix_layout, char norm, char uplo,
                      armpl_int_t n, const armpl_doublecomplex_t *a,
                      armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clansy*, *LAPACKE_dlansy*, *LAPACKE_slansy* and *LAPACKE_zlansy*. It also exists with a native Fortran interface as *zlansy*.

4.19.2140 LAPACKE_zlansy_work

Syntax

```
#include "armpl.h"

double LAPACKE_zlansy_work(armpl_int_t matrix_layout, char norm, char uplo,
                           armpl_int_t n, const armpl_doublecomplex_t *a,
                           armpl_int_t lda, double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clansy_work*, *LAPACKE_dlansy_work*, *LAPACKE_slansy_work* and *LAPACKE_zlansy_work*.

4.19.2141 LAPACKE_zlantr

Syntax

```
#include "armpl.h"

double LAPACKE_zlantr(armpl_int_t matrix_layout, char norm, char uplo,
                      char diag, armpl_int_t m, armpl_int_t n,
                      const armpl_doublecomplex_t *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clantr*, *LAPACKE_dlantr*, *LAPACKE_slantr* and *LAPACKE_zlantr*. It also exists with a native Fortran interface as *zlantr*.

4.19.2142 LAPACKE_zlantr_work

Syntax

```
#include "armpl.h"

double LAPACKE_zlantr_work(armpl_int_t matrix_layout, char norm, char uplo,
                           char diag, armpl_int_t m, armpl_int_t n,
                           const armpl_doublecomplex_t *a, armpl_int_t lda,
                           double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clantr_work*, *LAPACKE_dlantr_work*, *LAPACKE_slantr_work* and *LAPACKE_zlantr_work*.

4.19.2143 LAPACKE_zlapmr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlapmr(armpl_int_t matrix_layout, armpl_int_t forwrd,
                           armpl_int_t m, armpl_int_t n,
                           armpl_doublecomplex_t *x, armpl_int_t ldx,
                           armpl_int_t *k);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clapmr*, *LAPACKE_dlapmr*, *LAPACKE_slapmr* and *LAPACKE_zlapmr*. It also exists with a native Fortran interface as *zlapmr*.

4.19.2144 LAPACKE_zlapmr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlapmr_work(armpl_int_t matrix_layout, armpl_int_t forwrd,
                                armpl_int_t m, armpl_int_t n,
                                armpl_doublecomplex_t *x, armpl_int_t ldx,
                                armpl_int_t *k);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clapmr_work*, *LAPACKE_dlapmr_work*, *LAPACKE_slapmr_work* and *LAPACKE_zlapmr_work*.

4.19.2145 LAPACKE_zlapmt

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlapmt(armpl_int_t matrix_layout, armpl_int_t forwrd,
                           armpl_int_t m, armpl_int_t n,
                           armpl_doublecomplex_t *x, armpl_int_t ldx,
                           armpl_int_t *k);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clapmt*, *LAPACKE_dlapmt*, *LAPACKE_slapmt* and *LAPACKE_zlapmt*. It also exists with a native Fortran interface as *zlapmt*.

4.19.2146 LAPACKE_zlapmt_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlapmt_work(armpl_int_t matrix_layout, armpl_int_t forwrd,
                                armpl_int_t m, armpl_int_t n,
                                armpl_doublecomplex_t *x, armpl_int_t ldx,
                                armpl_int_t *k);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clapmt_work*, *LAPACKE_dlapmt_work*, *LAPACKE_slapmt_work* and *LAPACKE_zlapmt_work*.

4.19.2147 LAPACKE_zlarcm

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlarcm(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, const double *a, armpl_int_t lda,
                           const armpl_doublecomplex_t *b, armpl_int_t ldb,
                           armpl_doublecomplex_t *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarcm* and *LAPACKE_zlarcm*. It also exists with a native Fortran interface as *zlarcm*.

4.19.2148 LAPACKE_zlarcm_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlarcm_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, const double *a,
                                armpl_int_t lda,
                                const armpl_doublecomplex_t *b,
                                armpl_int_t ldb, armpl_doublecomplex_t *c,
                                armpl_int_t ldc, double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarcm_work* and *LAPACKE_zlarcm_work*.

4.19.2149 LAPACKE_zlarfb

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlarfb(armpl_int_t matrix_layout, char side, char trans,
                           char direct, char storev, armpl_int_t m,
                           armpl_int_t n, armpl_int_t k,
                           const armpl_doublecomplex_t *v, armpl_int_t ldv,
                           const armpl_doublecomplex_t *t, armpl_int_t ldt,
                           armpl_doublecomplex_t *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarfb*, *LAPACKE_dlarfb*, *LAPACKE_slarfb* and *LAPACKE_zlarfb*. It also exists with a native Fortran interface as *zlarfb*.

4.19.2150 LAPACKE_zlarfb_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlarfb_work(armpl_int_t matrix_layout, char side,
                                char trans, char direct, char storev,
                                armpl_int_t m, armpl_int_t n, armpl_int_t k,
                                const armpl_doublecomplex_t *v,
                                armpl_int_t ldv,
                                const armpl_doublecomplex_t *t,
                                armpl_int_t ldt, armpl_doublecomplex_t *c,
                                armpl_int_t ldc, armpl_doublecomplex_t *work,
                                armpl_int_t ldwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarfb_work*, *LAPACKE_dlarfb_work*, *LAPACKE_slarfb_work* and *LAPACKE_zlarfb_work*.

4.19.2151 LAPACKE_zlarfg

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlarfg(armpl_int_t n, armpl_doublecomplex_t *alpha,
                           armpl_doublecomplex_t *x, armpl_int_t incx,
                           armpl_doublecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarfg*, *LAPACKE_dlarfg*, *LAPACKE_slarfg* and *LAPACKE_zlarfg*. It also exists with a native Fortran interface as *zlarfg*.

4.19.2152 LAPACKE_zlarfg_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlarfg_work(armpl_int_t n, armpl_doublecomplex_t *alpha,
                                armpl_doublecomplex_t *x, armpl_int_t incx,
                                armpl_doublecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarfg_work*, *LAPACKE_dlarfg_work*, *LAPACKE_slarfg_work* and *LAPACKE_zlarfg_work*.

4.19.2153 LAPACKE_zlarft

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlarft(armpl_int_t matrix_layout, char direct,
                           char storev, armpl_int_t n, armpl_int_t k,
                           const armpl_doublecomplex_t *v, armpl_int_t ldv,
                           const armpl_doublecomplex_t *tau,
                           armpl_doublecomplex_t *t, armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarft*, *LAPACKE_dlarft*, *LAPACKE_slarft* and *LAPACKE_zlarft*. It also exists with a native Fortran interface as *zlarft*.

4.19.2154 LAPACKE_zlarft_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlarft_work(armpl_int_t matrix_layout, char direct,
                                char storev, armpl_int_t n, armpl_int_t k,
                                const armpl_doublecomplex_t *v,
                                armpl_int_t ldv,
                                const armpl_doublecomplex_t *tau,
                                armpl_doublecomplex_t *t, armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarft_work*, *LAPACKE_dlarft_work*, *LAPACKE_slarft_work* and *LAPACKE_zlarft_work*.

4.19.2155 LAPACKE_zlarfx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlarfx(armpl_int_t matrix_layout, char side,
                           armpl_int_t m, armpl_int_t n,
                           const armpl_doublecomplex_t *v,
                           armpl_doublecomplex_t tau,
                           armpl_doublecomplex_t *c, armpl_int_t ldc,
                           armpl_doublecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarfx*, *LAPACKE_dlarfx*, *LAPACKE_slarfx* and *LAPACKE_zlarfx*. It also exists with a native Fortran interface as *zlarfx*.

4.19.2156 LAPACKE_zlarfx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlarfx_work(armpl_int_t matrix_layout, char side,
                                armpl_int_t m, armpl_int_t n,
                                const armpl_doublecomplex_t *v,
                                armpl_doublecomplex_t tau,
                                armpl_doublecomplex_t *c, armpl_int_t ldc,
                                armpl_doublecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarfx_work*, *LAPACKE_dlarfx_work*, *LAPACKE_slarfx_work* and *LAPACKE_zlarfx_work*.

4.19.2157 LAPACKE_zlarnv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlarnv(armpl_int_t idist, armpl_int_t *iseed,
                           armpl_int_t n, armpl_doublecomplex_t *x);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarnv*, *LAPACKE_dlarnv*, *LAPACKE_slarnv* and *LAPACKE_zlarnv*. It also exists with a native Fortran interface as *zlarnv*.

4.19.2158 LAPACKE_zlarnv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlarnv_work(armpl_int_t idist, armpl_int_t *iseed,
                                armpl_int_t n, armpl_doublecomplex_t *x);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clarnv_work*, *LAPACKE_dlarnv_work*, *LAPACKE_slarnv_work* and *LAPACKE_zlarnv_work*.

4.19.2159 LAPACKE_zlascl

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlascl(armpl_int_t matrix_layout, char type,
                           armpl_int_t kl, armpl_int_t ku, double cfrom,
                           double cto, armpl_int_t m, armpl_int_t n,
                           armpl_doublecomplex_t *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clascl*, *LAPACKE_dlascl*, *LAPACKE_slascl* and *LAPACKE_zlascl*. It also exists with a native Fortran interface as *zlascl*.

4.19.2160 LAPACKE_zlascl_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlascl_work(armpl_int_t matrix_layout, char type,
                                armpl_int_t kl, armpl_int_t ku, double cfrom,
                                double cto, armpl_int_t m, armpl_int_t n,
                                armpl_doublecomplex_t *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clascl_work*, *LAPACKE_dlascl_work*, *LAPACKE_slascl_work* and *LAPACKE_zlascl_work*.

4.19.2161 LAPACKE_zlaset

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlaset(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t m, armpl_int_t n,
                           armpl_doublecomplex_t alpha,
                           armpl_doublecomplex_t beta,
                           armpl_doublecomplex_t *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_claset*, *LAPACKE_dlaset*, *LAPACKE_slaset* and *LAPACKE_zlaset*. It also exists with a native Fortran interface as *zlaset*.

4.19.2162 LAPACKE_zlaset_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlaset_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t m, armpl_int_t n,
                                armpl_doublecomplex_t alpha,
                                armpl_doublecomplex_t beta,
                                armpl_doublecomplex_t *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_claset_work*, *LAPACKE_dlaset_work*, *LAPACKE_slaset_work* and *LAPACKE_zlaset_work*.

4.19.2163 LAPACKE_zlassq

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlassq(armpl_int_t n, armpl_doublecomplex_t *x,
                           armpl_int_t incx, double *scale, double *sumsq);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_classq*, *LAPACKE_dlassq*, *LAPACKE_slassq* and *LAPACKE_zlassq*. It also exists with a native Fortran interface as *zlassq*.

4.19.2164 LAPACKE_zlassq_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlassq_work(armpl_int_t n, armpl_doublecomplex_t *x,
                                armpl_int_t incx, double *scale,
                                double *sumsq);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_classq_work*, *LAPACKE_dlassq_work*, *LAPACKE_slassq_work* and *LAPACKE_zlassq_work*.

4.19.2165 LAPACKE_zlaswp

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlaswp(armpl_int_t matrix_layout, armpl_int_t n,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           armpl_int_t k1, armpl_int_t k2,
                           const armpl_int_t *ipiv, armpl_int_t incx);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_claswp*, *LAPACKE_dlaswp*, *LAPACKE_slaswp* and *LAPACKE_zlaswp*. It also exists with a native Fortran interface as *zlaswp*.

4.19.2166 LAPACKE_zlaswp_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlaswp_work(armpl_int_t matrix_layout, armpl_int_t n,
                                armpl_doublecomplex_t *a, armpl_int_t lda,
                                armpl_int_t k1, armpl_int_t k2,
                                const armpl_int_t *ipiv, armpl_int_t incx);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_claswp_work*, *LAPACKE_dlaswp_work*, *LAPACKE_slaswp_work* and *LAPACKE_zlaswp_work*.

4.19.2167 LAPACKE_zlatms

Syntax

Routine is deprecated. Please choose a suitable alternative.

4.19.2168 LAPACKE_zlatms_work

Syntax

Routine is deprecated. Please choose a suitable alternative.

4.19.2169 LAPACKE_zlauum

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlauum(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_doublecomplex_t *a,
                           armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clauum*, *LAPACKE_dlauum*, *LAPACKE_slauum* and *LAPACKE_zlauum*. It also exists with a native Fortran interface as *zlauum*.

4.19.2170 LAPACKE_zlauum_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zlauum_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_doublecomplex_t *a,
                                armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_clauum_work*, *LAPACKE_dlauum_work*, *LAPACKE_slauum_work* and *LAPACKE_zlauum_work*.

4.19.2171 LAPACKE_zpbcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpbcon(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t kd,
                           const armpl_doublecomplex_t *ab, armpl_int_t ldab,
                           double anorm, double *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbcon*, *LAPACKE_dpbcon*, *LAPACKE_spbcon* and *LAPACKE_zpbcon*. It also exists with a native Fortran interface as *zpbcon*.

4.19.2172 LAPACKE_zpbcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpbcon_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t kd,
                                const armpl_doublecomplex_t *ab,
                                armpl_int_t ldab, double anorm, double *rcond,
                                armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbcon_work*, *LAPACKE_dpbcon_work*, *LAPACKE_spbcon_work* and *LAPACKE_zpbcon_work*.

4.19.2173 LAPACKE_zpbequ

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpbequ(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t kd,
                           const armpl_doublecomplex_t *ab, armpl_int_t ldab,
                           double *s, double *scond, double *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbequ*, *LAPACKE_dpbequ*, *LAPACKE_spbequ* and *LAPACKE_zpbequ*. It also exists with a native Fortran interface as *zpbequ*.

4.19.2174 LAPACKE_zpbequ_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpbequ_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t kd,
                                const armpl_doublecomplex_t *ab,
                                armpl_int_t ldab, double *s, double *scond,
                                double *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbequ_work*, *LAPACKE_dpbequ_work*, *LAPACKE_spbequ_work* and *LAPACKE_zpbequ_work*.

4.19.2175 LAPACKE_zpbrfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpbrfs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t kd, armpl_int_t nrhs,
                           const armpl_doublecomplex_t *ab, armpl_int_t ldab,
                           const armpl_doublecomplex_t *afb,
                           armpl_int_t ldafb, const armpl_doublecomplex_t *b,
                           armpl_int_t ldb, armpl_doublecomplex_t *x,
                           armpl_int_t ldx, double *ferr, double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbrfs*, *LAPACKE_dpbrfs*, *LAPACKE_spbrfs* and *LAPACKE_zpbrfs*. It also exists with a native Fortran interface as *zpbrfs*.

4.19.2176 LAPACKE_zpbrfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpbrfs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t kd,
                                armpl_int_t nrhs,
                                const armpl_doublecomplex_t *ab,
                                armpl_int_t ldab,
                                const armpl_doublecomplex_t *afb,
                                armpl_int_t ldafb,
                                const armpl_doublecomplex_t *b,
                                armpl_int_t ldb, armpl_doublecomplex_t *x,
                                armpl_int_t ldx, double *ferr, double *berr,
                                armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbrfs_work*, *LAPACKE_dpbrfs_work*, *LAPACKE_spbrfs_work* and *LAPACKE_zpbrfs_work*.

4.19.2177 LAPACKE_zpbstf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpbstf(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t kb,
                           armpl_doublecomplex_t *bb, armpl_int_t ldbb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbstf*, *LAPACKE_dpbstf*, *LAPACKE_spbstf* and *LAPACKE_zpbstf*. It also exists with a native Fortran interface as *zpbstf*.

4.19.2178 LAPACKE_zpbstf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpbstf_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t kb,
                                armpl_doublecomplex_t *bb, armpl_int_t ldbb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbstf_work*, *LAPACKE_dpbstf_work*, *LAPACKE_spbstf_work* and *LAPACKE_zpbstf_work*.

4.19.2179 LAPACKE_zpbsv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpbsv(armpl_int_t matrix_layout, char uplo, armpl_int_t n,
                          armpl_int_t kd, armpl_int_t nrhs,
                          armpl_doublecomplex_t *ab, armpl_int_t ldab,
                          armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbsv*, *LAPACKE_dpbsv*, *LAPACKE_spbsv* and *LAPACKE_zpbsv*. It also exists with a native Fortran interface as *zpbsv*.

4.19.2180 LAPACKE_zpbsv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpbsv_work(armpl_int_t matrix_layout, char uplo,
                               armpl_int_t n, armpl_int_t kd,
                               armpl_int_t nrhs, armpl_doublecomplex_t *ab,
                               armpl_int_t ldab, armpl_doublecomplex_t *b,
                               armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbsv_work*, *LAPACKE_dpbsv_work*, *LAPACKE_spbsv_work* and *LAPACKE_zpbsv_work*.

4.19.2181 LAPACKE_zpbsvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpbsvx(armpl_int_t matrix_layout, char fact, char uplo,
                           armpl_int_t n, armpl_int_t kd, armpl_int_t nrhs,
                           armpl_doublecomplex_t *ab, armpl_int_t ldab,
                           armpl_doublecomplex_t *afb, armpl_int_t ldafb,
                           char *equed, double *s, armpl_doublecomplex_t *b,
                           armpl_int_t ldb, armpl_doublecomplex_t *x,
                           armpl_int_t ldx, double *rcond, double *ferr,
                           double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbsvx*, *LAPACKE_dpbsvx*, *LAPACKE_spbsvx* and *LAPACKE_zpbsvx*. It also exists with a native Fortran interface as *zpbsvx*.

4.19.2182 LAPACKE_zpbsvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpbsvx_work(armpl_int_t matrix_layout, char fact,
                                char uplo, armpl_int_t n, armpl_int_t kd,
                                armpl_int_t nrhs, armpl_doublecomplex_t *ab,
                                armpl_int_t ldab, armpl_doublecomplex_t *afb,
                                armpl_int_t ldafb, char *equed, double *s,
                                armpl_doublecomplex_t *b, armpl_int_t ldb,
                                armpl_doublecomplex_t *x, armpl_int_t ldx,
```

(continues on next page)

(continued from previous page)

```
                                double *rcond, double *ferr, double *berr,
                                armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbsvx_work*, *LAPACKE_dpbsvx_work*, *LAPACKE_spbsvx_work* and *LAPACKE_zpbsvx_work*.

4.19.2183 LAPACKE_zpbtrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpbtrf(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t kd,
                           armpl_doublecomplex_t *ab, armpl_int_t ldab);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbtrf*, *LAPACKE_dpbtrf*, *LAPACKE_spbtrf* and *LAPACKE_zpbtrf*. It also exists with a native Fortran interface as *zpbtrf*.

4.19.2184 LAPACKE_zpbtrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpbtrf_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t kd,
                                armpl_doublecomplex_t *ab, armpl_int_t ldab);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbtrf_work*, *LAPACKE_dpbtrf_work*, *LAPACKE_spbtrf_work* and *LAPACKE_zpbtrf_work*.

4.19.2185 LAPACKE_zpbtrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpbtrs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t kd, armpl_int_t nrhs,
                           const armpl_doublecomplex_t *ab, armpl_int_t ldab,
                           armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbtrs*, *LAPACKE_dpbtrs*, *LAPACKE_spbtrs* and *LAPACKE_zpbtrs*. It also exists with a native Fortran interface as *zpbtrs*.

4.19.2186 LAPACKE_zpbtrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpbtrs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t kd,
                                armpl_int_t nrhs,
                                const armpl_doublecomplex_t *ab,
                                armpl_int_t ldab, armpl_doublecomplex_t *b,
                                armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpbtrs_work*, *LAPACKE_dpbtrs_work*, *LAPACKE_spbtrs_work* and *LAPACKE_zpbtrs_work*.

4.19.2187 LAPACKE_zpftrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpftrf(armpl_int_t matrix_layout, char transr, char uplo,
                           armpl_int_t n, armpl_doublecomplex_t *a);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpftrf*, *LAPACKE_dpftrf*, *LAPACKE_spftrf* and *LAPACKE_zpftrf*. It also exists with a native Fortran interface as *zpftrf*.

4.19.2188 LAPACKE_zpftrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpftrf_work(armpl_int_t matrix_layout, char transr,
                                char uplo, armpl_int_t n,
                                armpl_doublecomplex_t *a);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpftrf_work*, *LAPACKE_dpftrf_work*, *LAPACKE_spftrf_work* and *LAPACKE_zpftrf_work*.

4.19.2189 LAPACKE_zpftri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpftri(armpl_int_t matrix_layout, char transr, char uplo,
                           armpl_int_t n, armpl_doublecomplex_t *a);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpftri*, *LAPACKE_dpftri*, *LAPACKE_spftri* and *LAPACKE_zpftri*. It also exists with a native Fortran interface as *zpftri*.

4.19.2190 LAPACKE_zpftri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpftri_work(armpl_int_t matrix_layout, char transr,
                                char uplo, armpl_int_t n,
                                armpl_doublecomplex_t *a);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpftri_work*, *LAPACKE_dpftri_work*, *LAPACKE_spftri_work* and *LAPACKE_zpftri_work*.

4.19.2191 LAPACKE_zpftrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpftrs(armpl_int_t matrix_layout, char transr, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_doublecomplex_t *a,
                           armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpftrs*, *LAPACKE_dpftrs*, *LAPACKE_spftrs* and *LAPACKE_zpftrs*. It also exists with a native Fortran interface as *zpftrs*.

4.19.2192 LAPACKE_zpftrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpftrs_work(armpl_int_t matrix_layout, char transr,
                                char uplo, armpl_int_t n, armpl_int_t nrhs,
                                const armpl_doublecomplex_t *a,
                                armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpftrs_work*, *LAPACKE_dpftrs_work*, *LAPACKE_spftrs_work* and *LAPACKE_zpftrs_work*.

4.19.2193 LAPACKE_zpocon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpocon(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, const armpl_doublecomplex_t *a,
                           armpl_int_t lda, double anorm, double *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.2194 LAPACKE_zpocon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpocon_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, const armpl_doublecomplex_t *a,
                                armpl_int_t lda, double anorm, double *rcond,
                                armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpocon_work*, *LAPACKE_dpocon_work*, *LAPACKE_spocon_work* and *LAPACKE_zpocon_work*.

4.19.2195 LAPACKE_zpoequ

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpoequ(armpl_int_t matrix_layout, armpl_int_t n,
                           const armpl_doublecomplex_t *a, armpl_int_t lda,
                           double *s, double *scond, double *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpoequ*, *LAPACKE_dpoequ*, *LAPACKE_spoequ* and *LAPACKE_zpoequ*. It also exists with a native Fortran interface as *zpoequ*.

4.19.2196 LAPACKE_zpoequ_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpoequ_work(armpl_int_t matrix_layout, armpl_int_t n,
                                const armpl_doublecomplex_t *a,
                                armpl_int_t lda, double *s, double *scond,
                                double *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpoequ_work*, *LAPACKE_dpoequ_work*, *LAPACKE_spoequ_work* and *LAPACKE_zpoequ_work*.

4.19.2197 LAPACKE_zpoequb

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpoequb(armpl_int_t matrix_layout, armpl_int_t n,
                            const armpl_doublecomplex_t *a, armpl_int_t lda,
                            double *s, double *scond, double *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpoequb*, *LAPACKE_dpoequb*, *LAPACKE_spoequb* and *LAPACKE_zpoequb*. It also exists with a native Fortran interface as *zpoequb*.

4.19.2198 LAPACKE_zpoequb_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpoequb_work(armpl_int_t matrix_layout, armpl_int_t n,
                                 const armpl_doublecomplex_t *a,
                                 armpl_int_t lda, double *s, double *scond,
                                 double *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpoequb_work*, *LAPACKE_dpoequb_work*, *LAPACKE_spoequb_work* and *LAPACKE_zpoequb_work*.

4.19.2199 LAPACKE_zporfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zporfs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_doublecomplex_t *a, armpl_int_t lda,
                           const armpl_doublecomplex_t *af, armpl_int_t ldaf,
                           const armpl_doublecomplex_t *b, armpl_int_t ldb,
                           armpl_doublecomplex_t *x, armpl_int_t ldx,
                           double *ferr, double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cporfs*, *LAPACKE_dporfs*, *LAPACKE_sporfs* and *LAPACKE_zporfs*. It also exists with a native Fortran interface as *zporfs*.

4.19.2200 LAPACKE_zporfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zporfs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const armpl_doublecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_doublecomplex_t *af,
                                armpl_int_t ldaf,
                                const armpl_doublecomplex_t *b,
                                armpl_int_t ldb, armpl_doublecomplex_t *x,
                                armpl_int_t ldx, double *ferr, double *berr,
                                armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cporfs_work*, *LAPACKE_dporfs_work*, *LAPACKE_sporfs_work* and *LAPACKE_zporfs_work*.

4.19.2201 LAPACKE_zporfsx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zporfsx(armpl_int_t matrix_layout, char uplo, char equed,
                            armpl_int_t n, armpl_int_t nrhs,
                            const armpl_doublecomplex_t *a, armpl_int_t lda,
                            const armpl_doublecomplex_t *af, armpl_int_t ldaf,
                            const double *s, const armpl_doublecomplex_t *b,
                            armpl_int_t ldb, armpl_doublecomplex_t *x,
                            armpl_int_t ldx, double *rcond, double *berr,
                            armpl_int_t n_err_bnds, double *err_bnds_norm,
                            double *err_bnds_comp, armpl_int_t nparams,
                            double *params);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cporfsx*, *LAPACKE_dporfsx*, *LAPACKE_sporfsx* and *LAPACKE_zporfsx*. It also exists with a native Fortran interface as *zporfsx*.

4.19.2202 LAPACKE_zporfsx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zporfsx_work(armpl_int_t matrix_layout, char uplo,
                                 char equed, armpl_int_t n, armpl_int_t nrhs,
                                 const armpl_doublecomplex_t *a,
                                 armpl_int_t lda,
                                 const armpl_doublecomplex_t *af,
                                 armpl_int_t ldaf, const double *s,
                                 const armpl_doublecomplex_t *b,
                                 armpl_int_t ldb, armpl_doublecomplex_t *x,
                                 armpl_int_t ldx, double *rcond, double *berr,
                                 armpl_int_t n_err_bnds,
                                 double *err_bnds_norm, double *err_bnds_comp,
                                 armpl_int_t nparams, double *params,
                                 armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cporfsx_work*, *LAPACKE_dporfsx_work*, *LAPACKE_sporfsx_work* and *LAPACKE_zporfsx_work*.

4.19.2203 LAPACKE_zposv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zposv(armpl_int_t matrix_layout, char uplo, armpl_int_t n,
                          armpl_int_t nrhs, armpl_doublecomplex_t *a,
                          armpl_int_t lda, armpl_doublecomplex_t *b,
                          armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cposv*, *LAPACKE_dposv*, *LAPACKE_sposv* and *LAPACKE_zposv*. It also exists with a native Fortran interface as *zposv*.

4.19.2204 LAPACKE_zposv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zposv_work(armpl_int_t matrix_layout, char uplo,
                               armpl_int_t n, armpl_int_t nrhs,
                               armpl_doublecomplex_t *a, armpl_int_t lda,
                               armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cposv_work*, *LAPACKE_dposv_work*, *LAPACKE_sposv_work* and *LAPACKE_zposv_work*.

4.19.2205 LAPACKE_zposvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zposvx(armpl_int_t matrix_layout, char fact, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           armpl_doublecomplex_t *af, armpl_int_t ldaf,
                           char *equed, double *s, armpl_doublecomplex_t *b,
                           armpl_int_t ldb, armpl_doublecomplex_t *x,
                           armpl_int_t ldx, double *rcond, double *ferr,
                           double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cposvx*, *LAPACKE_dposvx*, *LAPACKE_sposvx* and *LAPACKE_zposvx*. It also exists with a native Fortran interface as *zposvx*.

4.19.2206 LAPACKE_zposvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zposvx_work(armpl_int_t matrix_layout, char fact,
                                char uplo, armpl_int_t n, armpl_int_t nrhs,
                                armpl_doublecomplex_t *a, armpl_int_t lda,
                                armpl_doublecomplex_t *af, armpl_int_t ldaf,
                                char *equed, double *s,
                                armpl_doublecomplex_t *b, armpl_int_t ldb,
                                armpl_doublecomplex_t *x, armpl_int_t ldx,
                                double *rcond, double *ferr, double *berr,
                                armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cposvx_work*, *LAPACKE_dposvx_work*, *LAPACKE_sposvx_work* and *LAPACKE_zposvx_work*.

4.19.2207 LAPACKE_zposvxx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zposvxx(armpl_int_t matrix_layout, char fact, char uplo,
                            armpl_int_t n, armpl_int_t nrhs,
                            armpl_doublecomplex_t *a, armpl_int_t lda,
                            armpl_doublecomplex_t *af, armpl_int_t ldaf,
                            char *equed, double *s, armpl_doublecomplex_t *b,
                            armpl_int_t ldb, armpl_doublecomplex_t *x,
                            armpl_int_t ldx, double *rcond, double *rpvgrw,
                            double *berr, armpl_int_t n_err_bnds,
                            double *err_bnds_norm, double *err_bnds_comp,
                            armpl_int_t nparams, double *params);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cposvxx*, *LAPACKE_dposvxx*, *LAPACKE_sposvxx* and *LAPACKE_zposvxx*. It also exists with a native Fortran interface as *zposvxx*.

4.19.2208 LAPACKE_zposvxx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zposvxx_work(armpl_int_t matrix_layout, char fact,
                                 char uplo, armpl_int_t n, armpl_int_t nrhs,
                                 armpl_doublecomplex_t *a, armpl_int_t lda,
                                 armpl_doublecomplex_t *af, armpl_int_t ldaf,
                                 char *equed, double *s,
                                 armpl_doublecomplex_t *b, armpl_int_t ldb,
```

(continues on next page)

(continued from previous page)

```
                                armpl_doublecomplex_t *x, armpl_int_t ldx,
                                double *rcond, double *rpvgrw, double *berr,
                                armpl_int_t n_err_bnds,
                                double *err_bnds_norm, double *err_bnds_comp,
                                armpl_int_t nparams, double *params,
                                armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cposvxx_work*, *LAPACKE_dposvxx_work*, *LAPACKE_sposvxx_work* and *LAPACKE_zposvxx_work*.

4.19.2209 LAPACKE_zpotrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpotrf(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_doublecomplex_t *a,
                           armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpotrf*, *LAPACKE_dpotrf*, *LAPACKE_spotrf* and *LAPACKE_zpotrf*. It also exists with a native Fortran interface as *zpotrf*.

4.19.2210 LAPACKE_zpotrf2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpotrf2(armpl_int_t matrix_layout, char uplo,
                            armpl_int_t n, armpl_doublecomplex_t *a,
                            armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpotrf2*, *LAPACKE_dpotrf2*, *LAPACKE_spotrf2* and *LAPACKE_zpotrf2*. It also exists with a native Fortran interface as *zpotrf2*.

4.19.2211 LAPACKE_zpotrf2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpotrf2_work(armpl_int_t matrix_layout, char uplo,
                                 armpl_int_t n, armpl_doublecomplex_t *a,
                                 armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpotrf2_work*, *LAPACKE_dpotrf2_work*, *LAPACKE_spotrf2_work* and *LAPACKE_zpotrf2_work*.

4.19.2212 LAPACKE_zpotrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpotrf_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_doublecomplex_t *a,
                                armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpotrf_work*, *LAPACKE_dpotrf_work*, *LAPACKE_spotrf_work* and *LAPACKE_zpotrf_work*.

4.19.2213 LAPACKE_zpotri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpotri(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_doublecomplex_t *a,
                           armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpotri*, *LAPACKE_dpotri*, *LAPACKE_spotri* and *LAPACKE_zpotri*. It also exists with a native Fortran interface as *zpotri*.

4.19.2214 LAPACKE_zpotri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpotri_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_doublecomplex_t *a,
                                armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpotri_work*, *LAPACKE_dpotri_work*, *LAPACKE_spotri_work* and *LAPACKE_zpotri_work*.

4.19.2215 LAPACKE_zpotrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpotrs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_doublecomplex_t *a, armpl_int_t lda,
                           armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpotrs*, *LAPACKE_dpotrs*, *LAPACKE_spotrs* and *LAPACKE_zpotrs*. It also exists with a native Fortran interface as *zpotrs*.

4.19.2216 LAPACKE_zpotrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpotrs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const armpl_doublecomplex_t *a,
                                armpl_int_t lda, armpl_doublecomplex_t *b,
                                armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpotrs_work*, *LAPACKE_dpotrs_work*, *LAPACKE_spotrs_work* and *LAPACKE_zpotrs_work*.

4.19.2217 LAPACKE_zppcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zppcon(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, const armpl_doublecomplex_t *ap,
                           double anorm, double *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cppcon*, *LAPACKE_dppcon*, *LAPACKE_sppcon* and *LAPACKE_zppcon*. It also exists with a native Fortran interface as *zppcon*.

4.19.2218 LAPACKE_zppcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zppcon_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n,
                                const armpl_doublecomplex_t *ap, double anorm,
                                double *rcond, armpl_doublecomplex_t *work,
                                double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cppcon_work*, *LAPACKE_dppcon_work*, *LAPACKE_sppcon_work* and *LAPACKE_zppcon_work*.

4.19.2219 LAPACKE_zppequ

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zppequ(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, const armpl_doublecomplex_t *ap,
                           double *s, double *scond, double *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cppequ*, *LAPACKE_dppequ*, *LAPACKE_sppequ* and *LAPACKE_zppequ*. It also exists with a native Fortran interface as *zppequ*.

4.19.2220 LAPACKE_zppequ_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zppequ_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n,
                                const armpl_doublecomplex_t *ap, double *s,
                                double *scond, double *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cppequ_work*, *LAPACKE_dppequ_work*, *LAPACKE_sppequ_work* and *LAPACKE_zppequ_work*.

4.19.2221 LAPACKE_zpprfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpprfs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_doublecomplex_t *ap,
                           const armpl_doublecomplex_t *afp,
                           const armpl_doublecomplex_t *b, armpl_int_t ldb,
                           armpl_doublecomplex_t *x, armpl_int_t ldx,
                           double *ferr, double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpprfs*, *LAPACKE_dpprfs*, *LAPACKE_spprfs* and *LAPACKE_zpprfs*. It also exists with a native Fortran interface as *zpprfs*.

4.19.2222 LAPACKE_zpprfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpprfs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const armpl_doublecomplex_t *ap,
                                const armpl_doublecomplex_t *afp,
                                const armpl_doublecomplex_t *b,
                                armpl_int_t ldb, armpl_doublecomplex_t *x,
                                armpl_int_t ldx, double *ferr, double *berr,
                                armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpprfs_work*, *LAPACKE_dpprfs_work*, *LAPACKE_spprfs_work* and *LAPACKE_zpprfs_work*.

4.19.2223 LAPACKE_zppsv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zppsv(armpl_int_t matrix_layout, char uplo, armpl_int_t n,
                          armpl_int_t nrhs, armpl_doublecomplex_t *ap,
                          armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cppsv*, *LAPACKE_dppsv*, *LAPACKE_sppsv* and *LAPACKE_zppsv*. It also exists with a native Fortran interface as *zppsv*.

4.19.2224 LAPACKE_zppsv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zppsv_work(armpl_int_t matrix_layout, char uplo,
                               armpl_int_t n, armpl_int_t nrhs,
                               armpl_doublecomplex_t *ap,
                               armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cppsv_work*, *LAPACKE_dppsv_work*, *LAPACKE_sppsv_work* and *LAPACKE_zppsv_work*.

4.19.2225 LAPACKE_zppsvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zppsvx(armpl_int_t matrix_layout, char fact, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           armpl_doublecomplex_t *ap,
                           armpl_doublecomplex_t *afp, char *equed, double *s,
                           armpl_doublecomplex_t *b, armpl_int_t ldb,
                           armpl_doublecomplex_t *x, armpl_int_t ldx,
                           double *rcond, double *ferr, double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cppsvx*, *LAPACKE_dppsvx*, *LAPACKE_sppsvx* and *LAPACKE_zppsvx*. It also exists with a native Fortran interface as *zppsvx*.

4.19.2226 LAPACKE_zppsvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zppsvx_work(armpl_int_t matrix_layout, char fact,
                                char uplo, armpl_int_t n, armpl_int_t nrhs,
                                armpl_doublecomplex_t *ap,
                                armpl_doublecomplex_t *afp, char *equed,
                                double *s, armpl_doublecomplex_t *b,
                                armpl_int_t ldb, armpl_doublecomplex_t *x,
                                armpl_int_t ldx, double *rcond, double *ferr,
                                double *berr, armpl_doublecomplex_t *work,
                                double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cppsvx_work*, *LAPACKE_dppsvx_work*, *LAPACKE_sppsvx_work* and *LAPACKE_zppsvx_work*.

4.19.2227 LAPACKE_zpptrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpptrf(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_doublecomplex_t *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpptrf*, *LAPACKE_dpptrf*, *LAPACKE_spptrf* and *LAPACKE_zpptrf*. It also exists with a native Fortran interface as *zpptrf*.

4.19.2228 LAPACKE_zpptrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpptrf_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_doublecomplex_t *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpptrf_work*, *LAPACKE_dpptrf_work*, *LAPACKE_spptrf_work* and *LAPACKE_zpptrf_work*.

4.19.2229 LAPACKE_zpptri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpptri(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_doublecomplex_t *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpptri*, *LAPACKE_dpptri*, *LAPACKE_spptri* and *LAPACKE_zpptri*. It also exists with a native Fortran interface as *zpptri*.

4.19.2230 LAPACKE_zpptri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpptri_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_doublecomplex_t *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpptri_work*, *LAPACKE_dpptri_work*, *LAPACKE_spptri_work* and *LAPACKE_zpptri_work*.

4.19.2231 LAPACKE_zpptrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpptrs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_doublecomplex_t *ap,
                           armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpptrs*, *LAPACKE_dpptrs*, *LAPACKE_spptrs* and *LAPACKE_zpptrs*. It also exists with a native Fortran interface as *zpptrs*.

4.19.2232 LAPACKE_zpptrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpptrs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const armpl_doublecomplex_t *ap,
                                armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpptrs_work*, *LAPACKE_dpptrs_work*, *LAPACKE_spptrs_work* and *LAPACKE_zpptrs_work*.

4.19.2233 LAPACKE_zpstrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpstrf(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_doublecomplex_t *a,
                           armpl_int_t lda, armpl_int_t *piv,
                           armpl_int_t *rank, double tol);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpstrf*, *LAPACKE_dpstrf*, *LAPACKE_spstrf* and *LAPACKE_zpstrf*. It also exists with a native Fortran interface as *zpstrf*.

4.19.2234 LAPACKE_zpstrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpstrf_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_doublecomplex_t *a,
                                armpl_int_t lda, armpl_int_t *piv,
                                armpl_int_t *rank, double tol, double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpstrf_work*, *LAPACKE_dpstrf_work*, *LAPACKE_spstrf_work* and *LAPACKE_zpstrf_work*.

4.19.2235 LAPACKE_zptcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zptcon(armpl_int_t n, const double *d,
                           const armpl_doublecomplex_t *e, double anorm,
                           double *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cptcon*, *LAPACKE_dptcon*, *LAPACKE_sptcon* and *LAPACKE_zptcon*. It also exists with a native Fortran interface as *zptcon*.

4.19.2236 LAPACKE_zptcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zptcon_work(armpl_int_t n, const double *d,
                                const armpl_doublecomplex_t *e, double anorm,
                                double *rcond, double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cptcon_work*, *LAPACKE_dptcon_work*, *LAPACKE_sptcon_work* and *LAPACKE_zptcon_work*.

4.19.2237 LAPACKE_zpteqr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpteqr(armpl_int_t matrix_layout, char compz,
                           armpl_int_t n, double *d, double *e,
                           armpl_doublecomplex_t *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpteqr*, *LAPACKE_dpteqr*, *LAPACKE_spteqr* and *LAPACKE_zpteqr*. It also exists with a native Fortran interface as *zpteqr*.

4.19.2238 LAPACKE_zpteqr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpteqr_work(armpl_int_t matrix_layout, char compz,
                                armpl_int_t n, double *d, double *e,
                                armpl_doublecomplex_t *z, armpl_int_t ldz,
                                double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpteqr_work*, *LAPACKE_dpteqr_work*, *LAPACKE_spteqr_work* and *LAPACKE_zpteqr_work*.

4.19.2239 LAPACKE_zptrfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zptrfs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs, const double *d,
                           const armpl_doublecomplex_t *e, const double *df,
                           const armpl_doublecomplex_t *ef,
                           const armpl_doublecomplex_t *b, armpl_int_t ldb,
                           armpl_doublecomplex_t *x, armpl_int_t ldx,
                           double *ferr, double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cptrfs*, *LAPACKE_dptrfs*, *LAPACKE_sptrfs* and *LAPACKE_zptrfs*. It also exists with a native Fortran interface as *zptrfs*.

4.19.2240 LAPACKE_zptrfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zptrfs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const double *d,
                                const armpl_doublecomplex_t *e,
                                const double *df,
                                const armpl_doublecomplex_t *ef,
                                const armpl_doublecomplex_t *b,
                                armpl_int_t ldb, armpl_doublecomplex_t *x,
                                armpl_int_t ldx, double *ferr, double *berr,
                                armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cptrfs_work*, *LAPACKE_dptrfs_work*, *LAPACKE_sptrfs_work* and *LAPACKE_zptrfs_work*.

4.19.2241 LAPACKE_zptsv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zptsv(armpl_int_t matrix_layout, armpl_int_t n,
                          armpl_int_t nrhs, double *d,
                          armpl_doublecomplex_t *e, armpl_doublecomplex_t *b,
                          armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cptsv*, *LAPACKE_dptsv*, *LAPACKE_sptsv* and *LAPACKE_zptsv*. It also exists with a native Fortran interface as *zptsv*.

4.19.2242 LAPACKE_zptsv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zptsv_work(armpl_int_t matrix_layout, armpl_int_t n,
                               armpl_int_t nrhs, double *d,
                               armpl_doublecomplex_t *e,
                               armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cptsv_work*, *LAPACKE_dptsv_work*, *LAPACKE_sptsv_work* and *LAPACKE_zptsv_work*.

4.19.2243 LAPACKE_zptsvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zptsvx(armpl_int_t matrix_layout, char fact,
                           armpl_int_t n, armpl_int_t nrhs, const double *d,
                           const armpl_doublecomplex_t *e, double *df,
                           armpl_doublecomplex_t *ef,
                           const armpl_doublecomplex_t *b, armpl_int_t ldb,
                           armpl_doublecomplex_t *x, armpl_int_t ldx,
                           double *rcond, double *ferr, double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cptsvx*, *LAPACKE_dptsvx*, *LAPACKE_sptsvx* and *LAPACKE_zptsvx*. It also exists with a native Fortran interface as *zptsvx*.

4.19.2244 LAPACKE_zptsvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zptsvx_work(armpl_int_t matrix_layout, char fact,
                                armpl_int_t n, armpl_int_t nrhs,
                                const double *d,
                                const armpl_doublecomplex_t *e, double *df,
                                armpl_doublecomplex_t *ef,
                                const armpl_doublecomplex_t *b,
                                armpl_int_t ldb, armpl_doublecomplex_t *x,
                                armpl_int_t ldx, double *rcond, double *ferr,
                                double *berr, armpl_doublecomplex_t *work,
                                double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cptsvx_work*, *LAPACKE_dptsvx_work*, *LAPACKE_sptsvx_work* and *LAPACKE_zptsvx_work*.

4.19.2245 LAPACKE_zpttrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpttrf(armpl_int_t n, double *d,
                           armpl_doublecomplex_t *e);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpttrf*, *LAPACKE_dpttrf*, *LAPACKE_spttrf* and *LAPACKE_zpttrf*. It also exists with a native Fortran interface as *zpttrf*.

4.19.2246 LAPACKE_zpttrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpttrf_work(armpl_int_t n, double *d,
                                armpl_doublecomplex_t *e);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpttrf_work*, *LAPACKE_dpttrf_work*, *LAPACKE_spttrf_work* and *LAPACKE_zpttrf_work*.

4.19.2247 LAPACKE_zpttrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpttrs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs, const double *d,
                           const armpl_doublecomplex_t *e,
                           armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpttrs*, *LAPACKE_dpttrs*, *LAPACKE_spttrs* and *LAPACKE_zpttrs*. It also exists with a native Fortran interface as *zpttrs*.

4.19.2248 LAPACKE_zpttrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zpttrs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const double *d,
                                const armpl_doublecomplex_t *e,
                                armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cpttrs_work*, *LAPACKE_dpttrs_work*, *LAPACKE_spttrs_work* and *LAPACKE_zpttrs_work*.

4.19.2249 LAPACKE_zspcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zspcon(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, const armpl_doublecomplex_t *ap,
                           const armpl_int_t *ipiv, double anorm,
                           double *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cspcon*, *LAPACKE_dspcon*, *LAPACKE_sspcon* and *LAPACKE_zspcon*. It also exists with a native Fortran interface as *zspcon*.

4.19.2250 LAPACKE_zspcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zspcon_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n,
                                const armpl_doublecomplex_t *ap,
                                const armpl_int_t *ipiv, double anorm,
                                double *rcond, armpl_doublecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cspcon_work*, *LAPACKE_dspcon_work*, *LAPACKE_sspcon_work* and *LAPACKE_zspcon_work*.

4.19.2251 LAPACKE_zsprfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsprfs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_doublecomplex_t *ap,
                           const armpl_doublecomplex_t *afp,
                           const armpl_int_t *ipiv,
                           const armpl_doublecomplex_t *b, armpl_int_t ldb,
                           armpl_doublecomplex_t *x, armpl_int_t ldx,
                           double *ferr, double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csprfs*, *LAPACKE_dsprfs*, *LAPACKE_ssprfs* and *LAPACKE_zsprfs*. It also exists with a native Fortran interface as *zsprfs*.

4.19.2252 LAPACKE_zsprfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsprfs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const armpl_doublecomplex_t *ap,
                                const armpl_doublecomplex_t *afp,
                                const armpl_int_t *ipiv,
                                const armpl_doublecomplex_t *b,
                                armpl_int_t ldb, armpl_doublecomplex_t *x,
                                armpl_int_t ldx, double *ferr, double *berr,
                                armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csprfs_work*, *LAPACKE_dsprfs_work*, *LAPACKE_ssprfs_work* and *LAPACKE_zsprfs_work*.

4.19.2253 LAPACKE_zspsv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zspsv(armpl_int_t matrix_layout, char uplo, armpl_int_t n,
                          armpl_int_t nrhs, armpl_doublecomplex_t *ap,
                          armpl_int_t *ipiv, armpl_doublecomplex_t *b,
                          armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cspsv*, *LAPACKE_dspsv*, *LAPACKE_sspsv* and *LAPACKE_zspsv*. It also exists with a native Fortran interface as *zspsv*.

4.19.2254 LAPACKE_zspsv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zspsv_work(armpl_int_t matrix_layout, char uplo,
                               armpl_int_t n, armpl_int_t nrhs,
                               armpl_doublecomplex_t *ap, armpl_int_t *ipiv,
                               armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cspsv_work*, *LAPACKE_dspsv_work*, *LAPACKE_sspsv_work* and *LAPACKE_zspsv_work*.

4.19.2255 LAPACKE_zspsvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zspsvx(armpl_int_t matrix_layout, char fact, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_doublecomplex_t *ap,
                           armpl_doublecomplex_t *afp, armpl_int_t *ipiv,
                           const armpl_doublecomplex_t *b, armpl_int_t ldb,
                           armpl_doublecomplex_t *x, armpl_int_t ldx,
                           double *rcond, double *ferr, double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cspsvx*, *LAPACKE_dspsvx*, *LAPACKE_sspsvx* and *LAPACKE_zspsvx*. It also exists with a native Fortran interface as *zspsvx*.

4.19.2256 LAPACKE_zspsvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zspsvx_work(armpl_int_t matrix_layout, char fact,
                                char uplo, armpl_int_t n, armpl_int_t nrhs,
                                const armpl_doublecomplex_t *ap,
                                armpl_doublecomplex_t *afp, armpl_int_t *ipiv,
                                const armpl_doublecomplex_t *b,
                                armpl_int_t ldb, armpl_doublecomplex_t *x,
                                armpl_int_t ldx, double *rcond, double *ferr,
                                double *berr, armpl_doublecomplex_t *work,
                                double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cspsvx_work*, *LAPACKE_dspsvx_work*, *LAPACKE_sspsvx_work* and *LAPACKE_zspsvx_work*.

4.19.2257 LAPACKE_zsptrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsptrf(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_doublecomplex_t *ap,
                           armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csptrf*, *LAPACKE_dsptrf*, *LAPACKE_ssptrf* and *LAPACKE_zsptrf*. It also exists with a native Fortran interface as *zsptrf*.

4.19.2258 LAPACKE_zsptrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsptrf_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_doublecomplex_t *ap,
                                armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csptrf_work*, *LAPACKE_dsptrf_work*, *LAPACKE_ssptrf_work* and *LAPACKE_zsptrf_work*.

4.19.2259 LAPACKE_zsptri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsptri(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_doublecomplex_t *ap,
                           const armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csptri*, *LAPACKE_dsptri*, *LAPACKE_ssptri* and *LAPACKE_zsptri*. It also exists with a native Fortran interface as *zsptri*.

4.19.2260 LAPACKE_zsptri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsptri_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_doublecomplex_t *ap,
                                const armpl_int_t *ipiv,
                                armpl_doublecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csptri_work*, *LAPACKE_dsptri_work*, *LAPACKE_ssptri_work* and *LAPACKE_zsptri_work*.

4.19.2261 LAPACKE_zsptrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsptrs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_doublecomplex_t *ap,
                           const armpl_int_t *ipiv, armpl_doublecomplex_t *b,
                           armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csptrs*, *LAPACKE_dsptrs*, *LAPACKE_ssptrs* and *LAPACKE_zsptrs*. It also exists with a native Fortran interface as *zsptrs*.

4.19.2262 LAPACKE_zsptrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsptrs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const armpl_doublecomplex_t *ap,
                                const armpl_int_t *ipiv,
                                armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csptrs_work*, *LAPACKE_dsptrs_work*, *LAPACKE_ssptrs_work* and *LAPACKE_zsptrs_work*.

4.19.2263 LAPACKE_zstedc

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zstedc(armpl_int_t matrix_layout, char compz,
                           armpl_int_t n, double *d, double *e,
                           armpl_doublecomplex_t *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cstedc*, *LAPACKE_dstedc*, *LAPACKE_sstedc* and *LAPACKE_zstedc*. It also exists with a native Fortran interface as *zstedc*.

4.19.2264 LAPACKE_zstedc_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zstedc_work(armpl_int_t matrix_layout, char compz,
                                armpl_int_t n, double *d, double *e,
                                armpl_doublecomplex_t *z, armpl_int_t ldz,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork, double *rwork,
                                armpl_int_t lrwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cstedc_work*, *LAPACKE_dstedc_work*, *LAPACKE_sstedc_work* and *LAPACKE_zstedc_work*.

4.19.2265 LAPACKE_zstegr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zstegr(armpl_int_t matrix_layout, char jobz, char range,
                           armpl_int_t n, double *d, double *e, double vl,
                           double vu, armpl_int_t il, armpl_int_t iu,
                           double abstol, armpl_int_t *m, double *w,
                           armpl_doublecomplex_t *z, armpl_int_t ldz,
                           armpl_int_t *isuppz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cstegr*, *LAPACKE_dstegr*, *LAPACKE_sstegr* and *LAPACKE_zstegr*. It also exists with a native Fortran interface as *zstegr*.

4.19.2266 LAPACKE_zstegr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zstegr_work(armpl_int_t matrix_layout, char jobz,
                                char range, armpl_int_t n, double *d,
                                double *e, double vl, double vu,
                                armpl_int_t il, armpl_int_t iu, double abstol,
                                armpl_int_t *m, double *w,
                                armpl_doublecomplex_t *z, armpl_int_t ldz,
                                armpl_int_t *isuppz, double *work,
                                armpl_int_t lwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cstegr_work*, *LAPACKE_dstegr_work*, *LAPACKE_sstegr_work* and *LAPACKE_zstegr_work*.

4.19.2267 LAPACKE_zstein

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zstein(armpl_int_t matrix_layout, armpl_int_t n,
                           const double *d, const double *e, armpl_int_t m,
                           const double *w, const armpl_int_t *iblock,
                           const armpl_int_t *isplit,
                           armpl_doublecomplex_t *z, armpl_int_t ldz,
                           armpl_int_t *ifailv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cstein*, *LAPACKE_dstein*, *LAPACKE_sstein* and *LAPACKE_zstein*. It also exists with a native Fortran interface as *zstein*.

4.19.2268 LAPACKE_zstein_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zstein_work(armpl_int_t matrix_layout, armpl_int_t n,
                                const double *d, const double *e,
                                armpl_int_t m, const double *w,
                                const armpl_int_t *iblock,
                                const armpl_int_t *isplit,
                                armpl_doublecomplex_t *z, armpl_int_t ldz,
                                double *work, armpl_int_t *iwork,
                                armpl_int_t *ifailv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cstein_work*, *LAPACKE_dstein_work*, *LAPACKE_sstein_work* and *LAPACKE_zstein_work*.

4.19.2269 LAPACKE_zstemr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zstemr(armpl_int_t matrix_layout, char jobz, char range,
                           armpl_int_t n, double *d, double *e, double vl,
                           double vu, armpl_int_t il, armpl_int_t iu,
                           armpl_int_t *m, double *w,
                           armpl_doublecomplex_t *z, armpl_int_t ldz,
                           armpl_int_t nzc, armpl_int_t *isuppz,
                           armpl_int_t *tryrac);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cstemr*, *LAPACKE_dstemr*, *LAPACKE_sstemr* and *LAPACKE_zstemr*. It also exists with a native Fortran interface as *zstemr*.

4.19.2270 LAPACKE_zstemr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zstemr_work(armpl_int_t matrix_layout, char jobz,
                                char range, armpl_int_t n, double *d,
                                double *e, double vl, double vu,
                                armpl_int_t il, armpl_int_t iu,
                                armpl_int_t *m, double *w,
                                armpl_doublecomplex_t *z, armpl_int_t ldz,
                                armpl_int_t nzc, armpl_int_t *isuppz,
                                armpl_int_t *tryrac, double *work,
                                armpl_int_t lwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cstemr_work*, *LAPACKE_dstemr_work*, *LAPACKE_sstemr_work* and *LAPACKE_zstemr_work*.

4.19.2271 LAPACKE_zsteqr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsteqr(armpl_int_t matrix_layout, char compz,
                           armpl_int_t n, double *d, double *e,
                           armpl_doublecomplex_t *z, armpl_int_t ldz);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csteqr*, *LAPACKE_dsteqr*, *LAPACKE_ssteqr* and *LAPACKE_zsteqr*. It also exists with a native Fortran interface as *zsteqr*.

4.19.2272 LAPACKE_zsteqr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsteqr_work(armpl_int_t matrix_layout, char compz,
                                armpl_int_t n, double *d, double *e,
                                armpl_doublecomplex_t *z, armpl_int_t ldz,
                                double *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csteqr_work*, *LAPACKE_dsteqr_work*, *LAPACKE_ssteqr_work* and *LAPACKE_zsteqr_work*.

4.19.2273 LAPACKE_zsycon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsycon(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, const armpl_doublecomplex_t *a,
                           armpl_int_t lda, const armpl_int_t *ipiv,
                           double anorm, double *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csycon*, *LAPACKE_dsycon*, *LAPACKE_ssycon* and *LAPACKE_zsycon*. It also exists with a native Fortran interface as *zsycon*.

4.19.2274 LAPACKE_zsycon_3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsycon_3(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, const armpl_doublecomplex_t *a,
                             armpl_int_t lda, const armpl_doublecomplex_t *e,
                             const armpl_int_t *ipiv, double anorm,
                             double *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csycon_3*, *LAPACKE_dsycon_3*, *LAPACKE_ssycon_3* and *LAPACKE_zsycon_3*. It also exists with a native Fortran interface as *zsycon_3*.

4.19.2275 LAPACKE_zsycon_3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsycon_3_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n,
                                  const armpl_doublecomplex_t *a,
                                  armpl_int_t lda,
                                  const armpl_doublecomplex_t *e,
                                  const armpl_int_t *ipiv, double anorm,
                                  double *rcond,
                                  armpl_doublecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csycon_3_work*, *LAPACKE_dsycon_3_work*, *LAPACKE_ssycon_3_work* and *LAPACKE_zsycon_3_work*.

4.19.2276 LAPACKE_zsycon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsycon_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, const armpl_doublecomplex_t *a,
                                armpl_int_t lda, const armpl_int_t *ipiv,
                                double anorm, double *rcond,
                                armpl_doublecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csycon_work*, *LAPACKE_dsycon_work*, *LAPACKE_ssycon_work* and *LAPACKE_zsycon_work*.

4.19.2277 LAPACKE_zsyconv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsyconv(armpl_int_t matrix_layout, char uplo, char way,
                            armpl_int_t n, armpl_doublecomplex_t *a,
                            armpl_int_t lda, const armpl_int_t *ipiv,
                            armpl_doublecomplex_t *e);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.2278 LAPACKE_zsyconv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsyconv_work(armpl_int_t matrix_layout, char uplo,
                                 char way, armpl_int_t n,
                                 armpl_doublecomplex_t *a, armpl_int_t lda,
                                 const armpl_int_t *ipiv,
                                 armpl_doublecomplex_t *e);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyconv_work*, *LAPACKE_dsyconv_work*, *LAPACKE_ssyconv_work* and *LAPACKE_zsyconv_work*.

4.19.2279 LAPACKE_zsyequb

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsyequb(armpl_int_t matrix_layout, char uplo,
                            armpl_int_t n, const armpl_doublecomplex_t *a,
                            armpl_int_t lda, double *s, double *scond,
                            double *amax);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyequb*, *LAPACKE_dsyequb*, *LAPACKE_ssyequb* and *LAPACKE_zsyequb*. It also exists with a native Fortran interface as *zsyequb*.

4.19.2280 LAPACKE_zsyequb_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsyequb_work(armpl_int_t matrix_layout, char uplo,
                                 armpl_int_t n,
                                 const armpl_doublecomplex_t *a,
                                 armpl_int_t lda, double *s, double *scond,
                                 double *amax, armpl_doublecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyequb_work*, *LAPACKE_dsyequb_work*, *LAPACKE_ssyequb_work* and *LAPACKE_zsyequb_work*.

4.19.2281 LAPACKE_zsyr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsyr(armpl_int_t matrix_layout, char uplo, armpl_int_t n,
                         armpl_doublecomplex_t alpha,
                         const armpl_doublecomplex_t *x, armpl_int_t incx,
                         armpl_doublecomplex_t *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyr* and *LAPACKE_zsyr*. It also exists with a native Fortran interface as *zsyr*.

4.19.2282 LAPACKE_zsyr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsyr_work(armpl_int_t matrix_layout, char uplo,
                              armpl_int_t n, armpl_doublecomplex_t alpha,
                              const armpl_doublecomplex_t *x,
                              armpl_int_t incx, armpl_doublecomplex_t *a,
                              armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyr_work* and *LAPACKE_zsyr_work*.

4.19.2283 LAPACKE_zsyrfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsyrfs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_doublecomplex_t *a, armpl_int_t lda,
                           const armpl_doublecomplex_t *af, armpl_int_t ldaf,
                           const armpl_int_t *ipiv,
                           const armpl_doublecomplex_t *b, armpl_int_t ldb,
                           armpl_doublecomplex_t *x, armpl_int_t ldx,
                           double *ferr, double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyrfs*, *LAPACKE_dsyrfs*, *LAPACKE_ssyrfs* and *LAPACKE_zsyrfs*. It also exists with a native Fortran interface as *zsyrfs*.

4.19.2284 LAPACKE_zsyrfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsyrfs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
```

(continues on next page)

(continued from previous page)

```
                                const armpl_doublecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_doublecomplex_t *af,
                                armpl_int_t ldaf, const armpl_int_t *ipiv,
                                const armpl_doublecomplex_t *b,
                                armpl_int_t ldb, armpl_doublecomplex_t *x,
                                armpl_int_t ldx, double *ferr, double *berr,
                                armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyrfs_work*, *LAPACKE_dsyrfs_work*, *LAPACKE_ssyrfs_work* and *LAPACKE_zsyrfs_work*.

4.19.2285 LAPACKE_zsyrfsx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsyrfsx(armpl_int_t matrix_layout, char uplo, char equed,
                            armpl_int_t n, armpl_int_t nrhs,
                            const armpl_doublecomplex_t *a, armpl_int_t lda,
                            const armpl_doublecomplex_t *af, armpl_int_t ldaf,
                            const armpl_int_t *ipiv, const double *s,
                            const armpl_doublecomplex_t *b, armpl_int_t ldb,
                            armpl_doublecomplex_t *x, armpl_int_t ldx,
                            double *rcond, double *berr,
                            armpl_int_t n_err_bnds, double *err_bnds_norm,
                            double *err_bnds_comp, armpl_int_t nparams,
                            double *params);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyrfsx*, *LAPACKE_dsyrfsx*, *LAPACKE_ssyrfsx* and *LAPACKE_zsyrfsx*. It also exists with a native Fortran interface as *zsyrfsx*.

4.19.2286 LAPACKE_zsyrfsx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsyrfsx_work(armpl_int_t matrix_layout, char uplo,
                                 char equed, armpl_int_t n, armpl_int_t nrhs,
                                 const armpl_doublecomplex_t *a,
                                 armpl_int_t lda,
                                 const armpl_doublecomplex_t *af,
                                 armpl_int_t ldaf, const armpl_int_t *ipiv,
                                 const double *s,
                                 const armpl_doublecomplex_t *b,
                                 armpl_int_t ldb, armpl_doublecomplex_t *x,
                                 armpl_int_t ldx, double *rcond, double *berr,
                                 armpl_int_t n_err_bnds,
                                 double *err_bnds_norm, double *err_bnds_comp,
                                 armpl_int_t nparams, double *params,
                                 armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyrfsx_work*, *LAPACKE_dsyrfsx_work*, *LAPACKE_ssyrfsx_work* and *LAPACKE_zsyrfsx_work*.

4.19.2287 LAPACKE_zsysv

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsysv(armpl_int_t matrix_layout, char uplo, armpl_int_t n,
                          armpl_int_t nrhs, armpl_doublecomplex_t *a,
                          armpl_int_t lda, armpl_int_t *ipiv,
                          armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysv*, *LAPACKE_dsysv*, *LAPACKE_ssysv* and *LAPACKE_zsysv*. It also exists with a native Fortran interface as *zsysv*.

4.19.2288 LAPACKE_zsysv_aa

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsysv_aa(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, armpl_int_t nrhs,
                             armpl_doublecomplex_t *a, armpl_int_t lda,
                             armpl_int_t *ipiv, armpl_doublecomplex_t *b,
                             armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysv_aa*, *LAPACKE_dsysv_aa*, *LAPACKE_ssysv_aa* and *LAPACKE_zsysv_aa*. It also exists with a native Fortran interface as *zsysv_aa*.

4.19.2289 LAPACKE_zsysv_aa_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsysv_aa_2stage(armpl_int_t matrix_layout, char uplo,
                                    armpl_int_t n, armpl_int_t nrhs,
                                    armpl_doublecomplex_t *a, armpl_int_t lda,
                                    armpl_doublecomplex_t *tb,
```

(continues on next page)

(continued from previous page)

```
                                    armpl_int_t ltb, armpl_int_t *ipiv,
                                    armpl_int_t *ipiv2,
                                    armpl_doublecomplex_t *b,
                                    armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysv_aa_2stage*, *LAPACKE_dsysv_aa_2stage*, *LAPACKE_ssysv_aa_2stage* and *LAPACKE_zsysv_aa_2stage*. It also exists with a native Fortran interface as *zsysv_aa_2stage*.

4.19.2290 LAPACKE_zsysv_aa_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsysv_aa_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, armpl_int_t nrhs,
                                  armpl_doublecomplex_t *a, armpl_int_t lda,
                                  armpl_int_t *ipiv, armpl_doublecomplex_t *b,
                                  armpl_int_t ldb,
                                  armpl_doublecomplex_t *work,
                                  armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysv_aa_work*, *LAPACKE_dsysv_aa_work*, *LAPACKE_ssysv_aa_work* and *LAPACKE_zsysv_aa_work*.

4.19.2291 LAPACKE_zsysv_rk

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsysv_rk(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, armpl_int_t nrhs,
                             armpl_doublecomplex_t *a, armpl_int_t lda,
                             armpl_doublecomplex_t *e, armpl_int_t *ipiv,
                             armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysv_rk*, *LAPACKE_dsysv_rk*, *LAPACKE_ssysv_rk* and *LAPACKE_zsysv_rk*. It also exists with a native Fortran interface as *zsysv_rk*.

4.19.2292 LAPACKE_zsysv_rk_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsysv_rk_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, armpl_int_t nrhs,
                                  armpl_doublecomplex_t *a, armpl_int_t lda,
                                  armpl_doublecomplex_t *e, armpl_int_t *ipiv,
                                  armpl_doublecomplex_t *b, armpl_int_t ldb,
                                  armpl_doublecomplex_t *work,
                                  armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysv_rk_work*, *LAPACKE_dsysv_rk_work*, *LAPACKE_ssysv_rk_work* and *LAPACKE_zsysv_rk_work*.

4.19.2293 LAPACKE_zsysv_rook

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsysv_rook(armpl_int_t matrix_layout, char uplo,
                               armpl_int_t n, armpl_int_t nrhs,
                               armpl_doublecomplex_t *a, armpl_int_t lda,
                               armpl_int_t *ipiv, armpl_doublecomplex_t *b,
                               armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysv_rook*, *LAPACKE_dsysv_rook*, *LAPACKE_ssysv_rook* and *LAPACKE_zsysv_rook*. It also exists with a native Fortran interface as *zsysv_rook*.

4.19.2294 LAPACKE_zsysv_rook_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsysv_rook_work(armpl_int_t matrix_layout, char uplo,
                                    armpl_int_t n, armpl_int_t nrhs,
                                    armpl_doublecomplex_t *a, armpl_int_t lda,
                                    armpl_int_t *ipiv,
                                    armpl_doublecomplex_t *b, armpl_int_t ldb,
                                    armpl_doublecomplex_t *work,
                                    armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysv_rook_work*, *LAPACKE_dsysv_rook_work*, *LAPACKE_ssysv_rook_work* and *LAPACKE_zsysv_rook_work*.

4.19.2295 LAPACKE_zsysv_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsysv_work(armpl_int_t matrix_layout, char uplo,
                               armpl_int_t n, armpl_int_t nrhs,
                               armpl_doublecomplex_t *a, armpl_int_t lda,
                               armpl_int_t *ipiv, armpl_doublecomplex_t *b,
                               armpl_int_t ldb, armpl_doublecomplex_t *work,
                               armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysv_work*, *LAPACKE_dsysv_work*, *LAPACKE_ssysv_work* and *LAPACKE_zsysv_work*.

4.19.2296 LAPACKE_zsysvx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsysvx(armpl_int_t matrix_layout, char fact, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_doublecomplex_t *a, armpl_int_t lda,
                           armpl_doublecomplex_t *af, armpl_int_t ldaf,
                           armpl_int_t *ipiv, const armpl_doublecomplex_t *b,
                           armpl_int_t ldb, armpl_doublecomplex_t *x,
                           armpl_int_t ldx, double *rcond, double *ferr,
                           double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysvx*, *LAPACKE_dsysvx*, *LAPACKE_ssysvx* and *LAPACKE_zsysvx*. It also exists with a native Fortran interface as *zsysvx*.

4.19.2297 LAPACKE_zsysvx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsysvx_work(armpl_int_t matrix_layout, char fact,
                                char uplo, armpl_int_t n, armpl_int_t nrhs,
                                const armpl_doublecomplex_t *a,
                                armpl_int_t lda, armpl_doublecomplex_t *af,
                                armpl_int_t ldaf, armpl_int_t *ipiv,
                                const armpl_doublecomplex_t *b,
                                armpl_int_t ldb, armpl_doublecomplex_t *x,
                                armpl_int_t ldx, double *rcond, double *ferr,
                                double *berr, armpl_doublecomplex_t *work,
                                armpl_int_t lwork, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysvx_work*, *LAPACKE_dsysvx_work*, *LAPACKE_ssysvx_work* and *LAPACKE_zsysvx_work*.

4.19.2298 LAPACKE_zsysvxx

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsysvxx(armpl_int_t matrix_layout, char fact, char uplo,
                            armpl_int_t n, armpl_int_t nrhs,
                            armpl_doublecomplex_t *a, armpl_int_t lda,
                            armpl_doublecomplex_t *af, armpl_int_t ldaf,
                            armpl_int_t *ipiv, char *equed, double *s,
                            armpl_doublecomplex_t *b, armpl_int_t ldb,
                            armpl_doublecomplex_t *x, armpl_int_t ldx,
                            double *rcond, double *rpvgrw, double *berr,
                            armpl_int_t n_err_bnds, double *err_bnds_norm,
                            double *err_bnds_comp, armpl_int_t nparams,
                            double *params);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysvxx*, *LAPACKE_dsysvxx*, *LAPACKE_ssysvxx* and *LAPACKE_zsysvxx*. It also exists with a native Fortran interface as *zsysvxx*.

4.19.2299 LAPACKE_zsysvxx_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsysvxx_work(armpl_int_t matrix_layout, char fact,
                                 char uplo, armpl_int_t n, armpl_int_t nrhs,
                                 armpl_doublecomplex_t *a, armpl_int_t lda,
                                 armpl_doublecomplex_t *af, armpl_int_t ldaf,
                                 armpl_int_t *ipiv, char *equed, double *s,
                                 armpl_doublecomplex_t *b, armpl_int_t ldb,
                                 armpl_doublecomplex_t *x, armpl_int_t ldx,
                                 double *rcond, double *rpvgrw, double *berr,
                                 armpl_int_t n_err_bnds,
                                 double *err_bnds_norm, double *err_bnds_comp,
                                 armpl_int_t nparams, double *params,
                                 armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csysvxx_work*, *LAPACKE_dsysvxx_work*, *LAPACKE_ssysvxx_work* and *LAPACKE_zsysvxx_work*.

4.19.2300 LAPACKE_zsyswapr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsyswapr(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, armpl_doublecomplex_t *a,
                             armpl_int_t lda, armpl_int_t i1,
                             armpl_int_t i2);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyswapr*, *LAPACKE_dsyswapr*, *LAPACKE_ssyswapr* and *LAPACKE_zsyswapr*. It also exists with a native Fortran interface as *zsyswapr*.

4.19.2301 LAPACKE_zsyswapr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsyswapr_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, armpl_doublecomplex_t *a,
                                  armpl_int_t lda, armpl_int_t i1,
                                  armpl_int_t i2);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csyswapr_work*, *LAPACKE_dsyswapr_work*, *LAPACKE_ssyswapr_work* and *LAPACKE_zsyswapr_work*.

4.19.2302 LAPACKE_zsytrf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsytrf(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_doublecomplex_t *a,
                           armpl_int_t lda, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrf*, *LAPACKE_dsytrf*, *LAPACKE_ssytrf* and *LAPACKE_zsytrf*. It also exists with a native Fortran interface as *zsytrf*.

4.19.2303 LAPACKE_zsytrf_aa

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsytrf_aa(armpl_int_t matrix_layout, char uplo,
                              armpl_int_t n, armpl_doublecomplex_t *a,
                              armpl_int_t lda, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.2304 LAPACKE_zsytrf_aa_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsytrf_aa_2stage(armpl_int_t matrix_layout, char uplo,
                                     armpl_int_t n, armpl_doublecomplex_t *a,
                                     armpl_int_t lda,
                                     armpl_doublecomplex_t *tb,
                                     armpl_int_t ltb, armpl_int_t *ipiv,
                                     armpl_int_t *ipiv2);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrf_aa_2stage*, *LAPACKE_dsytrf_aa_2stage*, *LAPACKE_ssytrf_aa_2stage* and *LAPACKE_zsytrf_aa_2stage*. It also exists with a native Fortran interface as *zsytrf_aa_2stage*.

4.19.2305 LAPACKE_zsytrf_aa_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsytrf_aa_work(armpl_int_t matrix_layout, char uplo,
                                   armpl_int_t n, armpl_doublecomplex_t *a,
                                   armpl_int_t lda, armpl_int_t *ipiv,
                                   armpl_doublecomplex_t *work,
                                   armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrf_aa_work*, *LAPACKE_dsytrf_aa_work*, *LAPACKE_ssytrf_aa_work* and *LAPACKE_zsytrf_aa_work*.

4.19.2306 LAPACKE_zsytrf_rk

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsytrf_rk(armpl_int_t matrix_layout, char uplo,
                              armpl_int_t n, armpl_doublecomplex_t *a,
                              armpl_int_t lda, armpl_doublecomplex_t *e,
                              armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrf_rk*, *LAPACKE_dsytrf_rk*, *LAPACKE_ssytrf_rk* and *LAPACKE_zsytrf_rk*. It also exists with a native Fortran interface as *zsytrf_rk*.

4.19.2307 LAPACKE_zsytrf_rk_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsytrf_rk_work(armpl_int_t matrix_layout, char uplo,
                                   armpl_int_t n, armpl_doublecomplex_t *a,
                                   armpl_int_t lda, armpl_doublecomplex_t *e,
                                   armpl_int_t *ipiv,
                                   armpl_doublecomplex_t *work,
                                   armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.2308 LAPACKE_zsytrf_rook

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsytrf_rook(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_doublecomplex_t *a,
                                armpl_int_t lda, armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrf_rook*, *LAPACKE_dsytrf_rook*, *LAPACKE_ssytrf_rook* and *LAPACKE_zsytrf_rook*. It also exists with a native Fortran interface as *zsytrf_rook*.

4.19.2309 LAPACKE_zsytrf_rook_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsytrf_rook_work(armpl_int_t matrix_layout, char uplo,
                                     armpl_int_t n, armpl_doublecomplex_t *a,
                                     armpl_int_t lda, armpl_int_t *ipiv,
                                     armpl_doublecomplex_t *work,
                                     armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrf_rook_work*, *LAPACKE_dsytrf_rook_work*, *LAPACKE_ssytrf_rook_work* and *LAPACKE_zsytrf_rook_work*.

4.19.2310 LAPACKE_zsytrf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsytrf_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_doublecomplex_t *a,
                                armpl_int_t lda, armpl_int_t *ipiv,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrf_work*, *LAPACKE_dsytrf_work*, *LAPACKE_ssytrf_work* and *LAPACKE_zsytrf_work*.

4.19.2311 LAPACKE_zsytri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsytri(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_doublecomplex_t *a,
                           armpl_int_t lda, const armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytri*, *LAPACKE_dsytri*, *LAPACKE_ssytri* and *LAPACKE_zsytri*. It also exists with a native Fortran interface as *zsytri*.

4.19.2312 LAPACKE_zsytri2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsytri2(armpl_int_t matrix_layout, char uplo,
                            armpl_int_t n, armpl_doublecomplex_t *a,
                            armpl_int_t lda, const armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytri2*, *LAPACKE_dsytri2*, *LAPACKE_ssytri2* and *LAPACKE_zsytri2*. It also exists with a native Fortran interface as *zsytri2*.

4.19.2313 LAPACKE_zsytri2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsytri2_work(armpl_int_t matrix_layout, char uplo,
                                 armpl_int_t n, armpl_doublecomplex_t *a,
                                 armpl_int_t lda, const armpl_int_t *ipiv,
                                 armpl_doublecomplex_t *work,
                                 armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytri2_work*, *LAPACKE_dsytri2_work*, *LAPACKE_ssytri2_work* and *LAPACKE_zsytri2_work*.

4.19.2314 LAPACKE_zsytri2x

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsytri2x(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, armpl_doublecomplex_t *a,
                             armpl_int_t lda, const armpl_int_t *ipiv,
                             armpl_int_t nb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytri2x*, *LAPACKE_dsytri2x*, *LAPACKE_ssytri2x* and *LAPACKE_zsytri2x*. It also exists with a native Fortran interface as *zsytri2x*.

4.19.2315 LAPACKE_zsytri2x_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsytri2x_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, armpl_doublecomplex_t *a,
                                  armpl_int_t lda, const armpl_int_t *ipiv,
                                  armpl_doublecomplex_t *work,
                                  armpl_int_t nb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytri2x_work*, *LAPACKE_dsytri2x_work*, *LAPACKE_ssytri2x_work* and *LAPACKE_zsytri2x_work*.

4.19.2316 LAPACKE_zsytri_3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsytri_3(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, armpl_doublecomplex_t *a,
                             armpl_int_t lda, const armpl_doublecomplex_t *e,
                             const armpl_int_t *ipiv);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytri_3*, *LAPACKE_dsytri_3*, *LAPACKE_ssytri_3* and *LAPACKE_zsytri_3*. It also exists with a native Fortran interface as *zsytri_3*.

4.19.2317 LAPACKE_zsytri_3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsytri_3_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, armpl_doublecomplex_t *a,
                                  armpl_int_t lda,
                                  const armpl_doublecomplex_t *e,
                                  const armpl_int_t *ipiv,
                                  armpl_doublecomplex_t *work,
                                  armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytri_3_work*, *LAPACKE_dsytri_3_work*, *LAPACKE_ssytri_3_work* and *LAPACKE_zsytri_3_work*.

4.19.2318 LAPACKE_zsytri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsytri_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_doublecomplex_t *a,
                                armpl_int_t lda, const armpl_int_t *ipiv,
                                armpl_doublecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytri_work*, *LAPACKE_dsytri_work*, *LAPACKE_ssytri_work* and *LAPACKE_zsytri_work*.

4.19.2319 LAPACKE_zsytrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsytrs(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_int_t nrhs,
                           const armpl_doublecomplex_t *a, armpl_int_t lda,
                           const armpl_int_t *ipiv, armpl_doublecomplex_t *b,
                           armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs*, *LAPACKE_dsytrs*, *LAPACKE_ssytrs* and *LAPACKE_zsytrs*. It also exists with a native Fortran interface as *zsytrs*.

4.19.2320 LAPACKE_zsytrs2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsytrs2(armpl_int_t matrix_layout, char uplo,
                            armpl_int_t n, armpl_int_t nrhs,
                            const armpl_doublecomplex_t *a, armpl_int_t lda,
                            const armpl_int_t *ipiv, armpl_doublecomplex_t *b,
                            armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs2*, *LAPACKE_dsytrs2*, *LAPACKE_ssytrs2* and *LAPACKE_zsytrs2*. It also exists with a native Fortran interface as *zsytrs2*.

4.19.2321 LAPACKE_zsytrs2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsytrs2_work(armpl_int_t matrix_layout, char uplo,
                                 armpl_int_t n, armpl_int_t nrhs,
                                 const armpl_doublecomplex_t *a,
                                 armpl_int_t lda, const armpl_int_t *ipiv,
```

(continues on next page)

(continued from previous page)

```
                                armpl_doublecomplex_t *b, armpl_int_t ldb,
                                armpl_doublecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs2_work*, *LAPACKE_dsytrs2_work*, *LAPACKE_ssytrs2_work* and *LAPACKE_zsytrs2_work*.

4.19.2322 LAPACKE_zsytrs_3

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsytrs_3(armpl_int_t matrix_layout, char uplo,
                             armpl_int_t n, armpl_int_t nrhs,
                             const armpl_doublecomplex_t *a, armpl_int_t lda,
                             const armpl_doublecomplex_t *e,
                             const armpl_int_t *ipiv,
                             armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs_3*, *LAPACKE_dsytrs_3*, *LAPACKE_ssytrs_3* and *LAPACKE_zsytrs_3*. It also exists with a native Fortran interface as *zsytrs_3*.

4.19.2323 LAPACKE_zsytrs_3_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsytrs_3_work(armpl_int_t matrix_layout, char uplo,
                                  armpl_int_t n, armpl_int_t nrhs,
                                  const armpl_doublecomplex_t *a,
                                  armpl_int_t lda,
                                  const armpl_doublecomplex_t *e,
                                  const armpl_int_t *ipiv,
                                  armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs_3_work*, *LAPACKE_dsytrs_3_work*, *LAPACKE_ssytrs_3_work* and *LAPACKE_zsytrs_3_work*.

4.19.2324 LAPACKE_zsytrs_aa

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsytrs_aa(armpl_int_t matrix_layout, char uplo,
                              armpl_int_t n, armpl_int_t nrhs,
                              const armpl_doublecomplex_t *a, armpl_int_t lda,
                              const armpl_int_t *ipiv,
                              armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs_aa*, *LAPACKE_dsytrs_aa*, *LAPACKE_ssytrs_aa* and *LAPACKE_zsytrs_aa*. It also exists with a native Fortran interface as *zsytrs_aa*.

4.19.2325 LAPACKE_zsytrs_aa_2stage

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsytrs_aa_2stage(armpl_int_t matrix_layout, char uplo,
                                     armpl_int_t n, armpl_int_t nrhs,
                                     armpl_doublecomplex_t *a,
                                     armpl_int_t lda,
                                     armpl_doublecomplex_t *tb,
                                     armpl_int_t ltb, armpl_int_t *ipiv,
                                     armpl_int_t *ipiv2,
                                     armpl_doublecomplex_t *b,
                                     armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs_aa_2stage*, *LAPACKE_dsytrs_aa_2stage*, *LAPACKE_ssytrs_aa_2stage* and *LAPACKE_zsytrs_aa_2stage*. It also exists with a native Fortran interface as *zsytrs_aa_2stage*.

4.19.2326 LAPACKE_zsytrs_aa_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsytrs_aa_work(armpl_int_t matrix_layout, char uplo,
                                   armpl_int_t n, armpl_int_t nrhs,
                                   const armpl_doublecomplex_t *a,
                                   armpl_int_t lda, const armpl_int_t *ipiv,
                                   armpl_doublecomplex_t *b, armpl_int_t ldb,
                                   armpl_doublecomplex_t *work,
                                   armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs_aa_work*, *LAPACKE_dsytrs_aa_work*, *LAPACKE_ssytrs_aa_work* and *LAPACKE_zsytrs_aa_work*.

4.19.2327 LAPACKE_zsytrs_rook

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsytrs_rook(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const armpl_doublecomplex_t *a,
                                armpl_int_t lda, const armpl_int_t *ipiv,
                                armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs_rook*, *LAPACKE_dsytrs_rook*, *LAPACKE_ssytrs_rook* and *LAPACKE_zsytrs_rook*. It also exists with a native Fortran interface as *zsytrs_rook*.

4.19.2328 LAPACKE_zsytrs_rook_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsytrs_rook_work(armpl_int_t matrix_layout, char uplo,
                                     armpl_int_t n, armpl_int_t nrhs,
                                     const armpl_doublecomplex_t *a,
                                     armpl_int_t lda, const armpl_int_t *ipiv,
```

(continues on next page)

(continued from previous page)

```
                                    armpl_doublecomplex_t *b,
                                    armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs_rook_work*, *LAPACKE_dsytrs_rook_work*, *LAPACKE_ssytrs_rook_work* and *LAPACKE_zsytrs_rook_work*.

4.19.2329 LAPACKE_zsytrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zsytrs_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_int_t nrhs,
                                const armpl_doublecomplex_t *a,
                                armpl_int_t lda, const armpl_int_t *ipiv,
                                armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_csytrs_work*, *LAPACKE_dsytrs_work*, *LAPACKE_ssytrs_work* and *LAPACKE_zsytrs_work*.

4.19.2330 LAPACKE_ztbcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztbcon(armpl_int_t matrix_layout, char norm, char uplo,
                           char diag, armpl_int_t n, armpl_int_t kd,
                           const armpl_doublecomplex_t *ab, armpl_int_t ldab,
                           double *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctbcon*, *LAPACKE_dtbcon*, *LAPACKE_stbcon* and *LAPACKE_ztbcon*. It also exists with a native Fortran interface as *ztbcon*.

4.19.2331 LAPACKE_ztbcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztbcon_work(armpl_int_t matrix_layout, char norm,
                                char uplo, char diag, armpl_int_t n,
                                armpl_int_t kd,
                                const armpl_doublecomplex_t *ab,
                                armpl_int_t ldab, double *rcond,
                                armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctbcon_work*, *LAPACKE_dtbcon_work*, *LAPACKE_stbcon_work* and *LAPACKE_ztbcon_work*.

4.19.2332 LAPACKE_ztbrfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztbrfs(armpl_int_t matrix_layout, char uplo, char trans,
                           char diag, armpl_int_t n, armpl_int_t kd,
                           armpl_int_t nrhs, const armpl_doublecomplex_t *ab,
                           armpl_int_t ldab, const armpl_doublecomplex_t *b,
                           armpl_int_t ldb, const armpl_doublecomplex_t *x,
                           armpl_int_t ldx, double *ferr, double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctbrfs*, *LAPACKE_dtbrfs*, *LAPACKE_stbrfs* and *LAPACKE_ztbrfs*. It also exists with a native Fortran interface as *ztbrfs*.

4.19.2333 LAPACKE_ztbrfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztbrfs_work(armpl_int_t matrix_layout, char uplo,
                                char trans, char diag, armpl_int_t n,
                                armpl_int_t kd, armpl_int_t nrhs,
                                const armpl_doublecomplex_t *ab,
                                armpl_int_t ldab,
                                const armpl_doublecomplex_t *b,
                                armpl_int_t ldb,
                                const armpl_doublecomplex_t *x,
                                armpl_int_t ldx, double *ferr, double *berr,
                                armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctbrfs_work*, *LAPACKE_dtbrfs_work*, *LAPACKE_stbrfs_work* and *LAPACKE_ztbrfs_work*.

4.19.2334 LAPACKE_ztbtrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztbtrs(armpl_int_t matrix_layout, char uplo, char trans,
                           char diag, armpl_int_t n, armpl_int_t kd,
                           armpl_int_t nrhs, const armpl_doublecomplex_t *ab,
                           armpl_int_t ldab, armpl_doublecomplex_t *b,
                           armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctbtrs*, *LAPACKE_dtbtrs*, *LAPACKE_stbtrs* and *LAPACKE_ztbtrs*. It also exists with a native Fortran interface as *ztbtrs*.

4.19.2335 LAPACKE_ztbtrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztbtrs_work(armpl_int_t matrix_layout, char uplo,
                                char trans, char diag, armpl_int_t n,
                                armpl_int_t kd, armpl_int_t nrhs,
                                const armpl_doublecomplex_t *ab,
                                armpl_int_t ldab, armpl_doublecomplex_t *b,
                                armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctbtrs_work*, *LAPACKE_dtbtrs_work*, *LAPACKE_stbtrs_work* and *LAPACKE_ztbtrs_work*.

4.19.2336 LAPACKE_ztfsm

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztfsm(armpl_int_t matrix_layout, char transr, char side,
                          char uplo, char trans, char diag, armpl_int_t m,
                          armpl_int_t n, armpl_doublecomplex_t alpha,
                          const armpl_doublecomplex_t *a,
                          armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctfsm*, *LAPACKE_dtfsm*, *LAPACKE_stfsm* and *LAPACKE_ztfsm*. It also exists with a native Fortran interface as *ztfsm*.

4.19.2337 LAPACKE_ztfsm_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztfsm_work(armpl_int_t matrix_layout, char transr,
                               char side, char uplo, char trans, char diag,
                               armpl_int_t m, armpl_int_t n,
                               armpl_doublecomplex_t alpha,
                               const armpl_doublecomplex_t *a,
                               armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctfsm_work*, *LAPACKE_dtfsm_work*, *LAPACKE_stfsm_work* and *LAPACKE_ztfsm_work*.

4.19.2338 LAPACKE_ztftri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztftri(armpl_int_t matrix_layout, char transr, char uplo,
                           char diag, armpl_int_t n,
                           armpl_doublecomplex_t *a);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctftri*, *LAPACKE_dtftri*, *LAPACKE_stftri* and *LAPACKE_ztftri*. It also exists with a native Fortran interface as *ztftri*.

4.19.2339 LAPACKE_ztftri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztftri_work(armpl_int_t matrix_layout, char transr,
                                char uplo, char diag, armpl_int_t n,
                                armpl_doublecomplex_t *a);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctftri_work*, *LAPACKE_dtftri_work*, *LAPACKE_stftri_work* and *LAPACKE_ztftri_work*.

4.19.2340 LAPACKE_ztfttp

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztfttp(armpl_int_t matrix_layout, char transr, char uplo,
                           armpl_int_t n, const armpl_doublecomplex_t *arf,
                           armpl_doublecomplex_t *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctfttp*, *LAPACKE_dtfttp*, *LAPACKE_stfttp* and *LAPACKE_ztfttp*. It also exists with a native Fortran interface as *ztfttp*.

4.19.2341 LAPACKE_ztfttp_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztfttp_work(armpl_int_t matrix_layout, char transr,
                                char uplo, armpl_int_t n,
                                const armpl_doublecomplex_t *arf,
                                armpl_doublecomplex_t *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctfttp_work*, *LAPACKE_dtfttp_work*, *LAPACKE_stfttp_work* and *LAPACKE_ztfttp_work*.

4.19.2342 LAPACKE_ztfttr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztfttr(armpl_int_t matrix_layout, char transr, char uplo,
                           armpl_int_t n, const armpl_doublecomplex_t *arf,
                           armpl_doublecomplex_t *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctfttr*, *LAPACKE_dtfttr*, *LAPACKE_stfttr* and *LAPACKE_ztfttr*. It also exists with a native Fortran interface as *ztfttr*.

4.19.2343 LAPACKE_ztfttr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztfttr_work(armpl_int_t matrix_layout, char transr,
                                char uplo, armpl_int_t n,
                                const armpl_doublecomplex_t *arf,
                                armpl_doublecomplex_t *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctfttr_work*, *LAPACKE_dtfttr_work*, *LAPACKE_stfttr_work* and *LAPACKE_ztfttr_work*.

4.19.2344 LAPACKE_ztgevc

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztgevc(armpl_int_t matrix_layout, char side, char howmny,
                           const armpl_int_t *select, armpl_int_t n,
                           const armpl_doublecomplex_t *s, armpl_int_t lds,
                           const armpl_doublecomplex_t *p, armpl_int_t ldp,
                           armpl_doublecomplex_t *vl, armpl_int_t ldvl,
                           armpl_doublecomplex_t *vr, armpl_int_t ldvr,
                           armpl_int_t mm, armpl_int_t *m);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgevc*, *LAPACKE_dtgevc*, *LAPACKE_stgevc* and *LAPACKE_ztgevc*. It also exists with a native Fortran interface as *ztgevc*.

4.19.2345 LAPACKE_ztgevc_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztgevc_work(armpl_int_t matrix_layout, char side,
                                char howmny, const armpl_int_t *select,
```

(continues on next page)

(continued from previous page)

```
                                armpl_int_t n, const armpl_doublecomplex_t *s,
                                armpl_int_t lds,
                                const armpl_doublecomplex_t *p,
                                armpl_int_t ldp, armpl_doublecomplex_t *vl,
                                armpl_int_t ldvl, armpl_doublecomplex_t *vr,
                                armpl_int_t ldvr, armpl_int_t mm,
                                armpl_int_t *m, armpl_doublecomplex_t *work,
                                double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgevc_work*, *LAPACKE_dtgevc_work*, *LAPACKE_stgevc_work* and *LAPACKE_ztgevc_work*.

4.19.2346 LAPACKE_ztgexc

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztgexc(armpl_int_t matrix_layout, armpl_int_t wantq,
                           armpl_int_t wantz, armpl_int_t n,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           armpl_doublecomplex_t *b, armpl_int_t ldb,
                           armpl_doublecomplex_t *q, armpl_int_t ldq,
                           armpl_doublecomplex_t *z, armpl_int_t ldz,
                           armpl_int_t ifst, armpl_int_t ilst);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgexc*, *LAPACKE_dtgexc*, *LAPACKE_stgexc* and *LAPACKE_ztgexc*. It also exists with a native Fortran interface as *ztgexc*.

4.19.2347 LAPACKE_ztgexc_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztgexc_work(armpl_int_t matrix_layout, armpl_int_t wantq,
                                armpl_int_t wantz, armpl_int_t n,
                                armpl_doublecomplex_t *a, armpl_int_t lda,
                                armpl_doublecomplex_t *b, armpl_int_t ldb,
                                armpl_doublecomplex_t *q, armpl_int_t ldq,
                                armpl_doublecomplex_t *z, armpl_int_t ldz,
                                armpl_int_t ifst, armpl_int_t ilst);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgexc_work*, *LAPACKE_dtgexc_work*, *LAPACKE_stgexc_work* and *LAPACKE_ztgexc_work*.

4.19.2348 LAPACKE_ztgsen

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztgsen(armpl_int_t matrix_layout, armpl_int_t ijob,
                           armpl_int_t wantq, armpl_int_t wantz,
                           const armpl_int_t *select, armpl_int_t n,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           armpl_doublecomplex_t *b, armpl_int_t ldb,
                           armpl_doublecomplex_t *alpha,
                           armpl_doublecomplex_t *beta,
                           armpl_doublecomplex_t *q, armpl_int_t ldq,
                           armpl_doublecomplex_t *z, armpl_int_t ldz,
                           armpl_int_t *m, double *pl, double *pr,
                           double *dif);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgsen*, *LAPACKE_dtgsen*, *LAPACKE_stgsen* and *LAPACKE_ztgsen*. It also exists with a native Fortran interface as *ztgsen*.

4.19.2349 LAPACKE_ztgsen_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztgsen_work(armpl_int_t matrix_layout, armpl_int_t ijob,
                                armpl_int_t wantq, armpl_int_t wantz,
                                const armpl_int_t *select, armpl_int_t n,
                                armpl_doublecomplex_t *a, armpl_int_t lda,
                                armpl_doublecomplex_t *b, armpl_int_t ldb,
                                armpl_doublecomplex_t *alpha,
                                armpl_doublecomplex_t *beta,
                                armpl_doublecomplex_t *q, armpl_int_t ldq,
                                armpl_doublecomplex_t *z, armpl_int_t ldz,
                                armpl_int_t *m, double *pl, double *pr,
                                double *dif, armpl_doublecomplex_t *work,
                                armpl_int_t lwork, armpl_int_t *iwork,
                                armpl_int_t liwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

4.19.2350 LAPACKE_ztgsja

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztgsja(armpl_int_t matrix_layout, char jobu, char jobv,
                           char jobq, armpl_int_t m, armpl_int_t p,
                           armpl_int_t n, armpl_int_t k, armpl_int_t l,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           armpl_doublecomplex_t *b, armpl_int_t ldb,
                           double tola, double tolb, double *alpha,
                           double *beta, armpl_doublecomplex_t *u,
```

(continues on next page)

(continued from previous page)

```
                                armpl_int_t ldu, armpl_doublecomplex_t *v,
                                armpl_int_t ldv, armpl_doublecomplex_t *q,
                                armpl_int_t ldq, armpl_int_t *ncycle);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgsja*, *LAPACKE_dtgsja*, *LAPACKE_stgsja* and *LAPACKE_ztgsja*. It also exists with a native Fortran interface as *ztgsja*.

4.19.2351 LAPACKE_ztgsja_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztgsja_work(armpl_int_t matrix_layout, char jobu,
                                char jobv, char jobq, armpl_int_t m,
                                armpl_int_t p, armpl_int_t n, armpl_int_t k,
                                armpl_int_t l, armpl_doublecomplex_t *a,
                                armpl_int_t lda, armpl_doublecomplex_t *b,
                                armpl_int_t ldb, double tola, double tolb,
                                double *alpha, double *beta,
                                armpl_doublecomplex_t *u, armpl_int_t ldu,
                                armpl_doublecomplex_t *v, armpl_int_t ldv,
                                armpl_doublecomplex_t *q, armpl_int_t ldq,
                                armpl_doublecomplex_t *work,
                                armpl_int_t *ncycle);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgsja_work*, *LAPACKE_dtgsja_work*, *LAPACKE_stgsja_work* and *LAPACKE_ztgsja_work*.

4.19.2352 LAPACKE_ztgsna

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztgsna(armpl_int_t matrix_layout, char job, char howmny,
                           const armpl_int_t *select, armpl_int_t n,
                           const armpl_doublecomplex_t *a, armpl_int_t lda,
                           const armpl_doublecomplex_t *b, armpl_int_t ldb,
                           const armpl_doublecomplex_t *vl, armpl_int_t ldvl,
                           const armpl_doublecomplex_t *vr, armpl_int_t ldvr,
                           double *s, double *dif, armpl_int_t mm,
                           armpl_int_t *m);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgsna*, *LAPACKE_dtgsna*, *LAPACKE_stgsna* and *LAPACKE_ztgsna*. It also exists with a native Fortran interface as *ztgsna*.

4.19.2353 LAPACKE_ztgsna_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztgsna_work(armpl_int_t matrix_layout, char job,
                                char howmny, const armpl_int_t *select,
                                armpl_int_t n, const armpl_doublecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_doublecomplex_t *b,
                                armpl_int_t ldb,
                                const armpl_doublecomplex_t *vl,
                                armpl_int_t ldvl,
                                const armpl_doublecomplex_t *vr,
                                armpl_int_t ldvr, double *s, double *dif,
                                armpl_int_t mm, armpl_int_t *m,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgsna_work*, *LAPACKE_dtgsna_work*, *LAPACKE_stgsna_work* and *LAPACKE_ztgsna_work*.

4.19.2354 LAPACKE_ztgsyl

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztgsyl(armpl_int_t matrix_layout, char trans,
                           armpl_int_t ijob, armpl_int_t m, armpl_int_t n,
                           const armpl_doublecomplex_t *a, armpl_int_t lda,
                           const armpl_doublecomplex_t *b, armpl_int_t ldb,
                           armpl_doublecomplex_t *c, armpl_int_t ldc,
                           const armpl_doublecomplex_t *d, armpl_int_t ldd,
                           const armpl_doublecomplex_t *e, armpl_int_t lde,
                           armpl_doublecomplex_t *f, armpl_int_t ldf,
                           double *scale, double *dif);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgsyl*, *LAPACKE_dtgsyl*, *LAPACKE_stgsyl* and *LAPACKE_ztgsyl*. It also exists with a native Fortran interface as *ztgsyl*.

4.19.2355 LAPACKE_ztgsyl_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztgsyl_work(armpl_int_t matrix_layout, char trans,
                                armpl_int_t ijob, armpl_int_t m,
                                armpl_int_t n, const armpl_doublecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_doublecomplex_t *b,
                                armpl_int_t ldb, armpl_doublecomplex_t *c,
```

(continues on next page)

(continued from previous page)

```
                                armpl_int_t ldc,
                                const armpl_doublecomplex_t *d,
                                armpl_int_t ldd,
                                const armpl_doublecomplex_t *e,
                                armpl_int_t lde, armpl_doublecomplex_t *f,
                                armpl_int_t ldf, double *scale, double *dif,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctgsyl_work*, *LAPACKE_dtgsyl_work*, *LAPACKE_stgsyl_work* and *LAPACKE_ztgsyl_work*.

4.19.2356 LAPACKE_ztpcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztpcon(armpl_int_t matrix_layout, char norm, char uplo,
                           char diag, armpl_int_t n,
                           const armpl_doublecomplex_t *ap, double *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpcon*, *LAPACKE_dtpcon*, *LAPACKE_stpcon* and *LAPACKE_ztpcon*. It also exists with a native Fortran interface as *ztpcon*.

4.19.2357 LAPACKE_ztpcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztpcon_work(armpl_int_t matrix_layout, char norm,
                                char uplo, char diag, armpl_int_t n,
                                const armpl_doublecomplex_t *ap,
                                double *rcond, armpl_doublecomplex_t *work,
                                double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpcon_work*, *LAPACKE_dtpcon_work*, *LAPACKE_stpcon_work* and *LAPACKE_ztpcon_work*.

4.19.2358 LAPACKE_ztpmqrt

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztpmqrt(armpl_int_t matrix_layout, char side, char trans,
                            armpl_int_t m, armpl_int_t n, armpl_int_t k,
                            armpl_int_t l, armpl_int_t nb,
                            const armpl_doublecomplex_t *v, armpl_int_t ldv,
                            const armpl_doublecomplex_t *t, armpl_int_t ldt,
                            armpl_doublecomplex_t *a, armpl_int_t lda,
                            armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpmqrt*, *LAPACKE_dtpmqrt*, *LAPACKE_stpmqrt* and *LAPACKE_ztpmqrt*. It also exists with a native Fortran interface as *ztpmqrt*.

4.19.2359 LAPACKE_ztpmqrt_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztpmqrt_work(armpl_int_t matrix_layout, char side,
                                 char trans, armpl_int_t m, armpl_int_t n,
                                 armpl_int_t k, armpl_int_t l, armpl_int_t nb,
                                 const armpl_doublecomplex_t *v,
                                 armpl_int_t ldv,
                                 const armpl_doublecomplex_t *t,
                                 armpl_int_t ldt, armpl_doublecomplex_t *a,
                                 armpl_int_t lda, armpl_doublecomplex_t *b,
                                 armpl_int_t ldb,
                                 armpl_doublecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpmqrt_work*, *LAPACKE_dtpmqrt_work*, *LAPACKE_stpmqrt_work* and *LAPACKE_ztpmqrt_work*.

4.19.2360 LAPACKE_ztpqrt

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztpqrt(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t l, armpl_int_t nb,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           armpl_doublecomplex_t *b, armpl_int_t ldb,
                           armpl_doublecomplex_t *t, armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpqrt*, *LAPACKE_dtpqrt*, *LAPACKE_stpqrt* and *LAPACKE_ztpqrt*. It also exists with a native Fortran interface as *ztpqrt*.

4.19.2361 LAPACKE_ztpqrt2

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztpqrt2(armpl_int_t matrix_layout, armpl_int_t m,
                            armpl_int_t n, armpl_int_t l,
                            armpl_doublecomplex_t *a, armpl_int_t lda,
                            armpl_doublecomplex_t *b, armpl_int_t ldb,
                            armpl_doublecomplex_t *t, armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpqrt2*, *LAPACKE_dtpqrt2*, *LAPACKE_stpqrt2* and *LAPACKE_ztpqrt2*. It also exists with a native Fortran interface as *ztpqrt2*.

4.19.2362 LAPACKE_ztpqrt2_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztpqrt2_work(armpl_int_t matrix_layout, armpl_int_t m,
                                 armpl_int_t n, armpl_int_t l,
                                 armpl_doublecomplex_t *a, armpl_int_t lda,
                                 armpl_doublecomplex_t *b, armpl_int_t ldb,
                                 armpl_doublecomplex_t *t, armpl_int_t ldt);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpqrt2_work*, *LAPACKE_dtpqrt2_work*, *LAPACKE_stpqrt2_work* and *LAPACKE_ztpqrt2_work*.

4.19.2363 LAPACKE_ztpqrt_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztpqrt_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t l, armpl_int_t nb,
                                armpl_doublecomplex_t *a, armpl_int_t lda,
                                armpl_doublecomplex_t *b, armpl_int_t ldb,
                                armpl_doublecomplex_t *t, armpl_int_t ldt,
                                armpl_doublecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpqrt_work*, *LAPACKE_dtpqrt_work*, *LAPACKE_stpqrt_work* and *LAPACKE_ztpqrt_work*.

4.19.2364 LAPACKE_ztprfb

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztprfb(armpl_int_t matrix_layout, char side, char trans,
                           char direct, char storev, armpl_int_t m,
                           armpl_int_t n, armpl_int_t k, armpl_int_t l,
```

(continues on next page)

(continued from previous page)

```
                            const armpl_doublecomplex_t *v, armpl_int_t ldv,
                            const armpl_doublecomplex_t *t, armpl_int_t ldt,
                            armpl_doublecomplex_t *a, armpl_int_t lda,
                            armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctprfb*, *LAPACKE_dtprfb*, *LAPACKE_stprfb* and *LAPACKE_ztprfb*. It also exists with a native Fortran interface as *ztprfb*.

4.19.2365 LAPACKE_ztprfb_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztprfb_work(armpl_int_t matrix_layout, char side,
                                char trans, char direct, char storev,
                                armpl_int_t m, armpl_int_t n, armpl_int_t k,
                                armpl_int_t l, const armpl_doublecomplex_t *v,
                                armpl_int_t ldv,
                                const armpl_doublecomplex_t *t,
                                armpl_int_t ldt, armpl_doublecomplex_t *a,
                                armpl_int_t lda, armpl_doublecomplex_t *b,
                                armpl_int_t ldb, armpl_doublecomplex_t *work,
                                armpl_int_t ldwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctprfb_work*, *LAPACKE_dtprfb_work*, *LAPACKE_stprfb_work* and *LAPACKE_ztprfb_work*.

4.19.2366 LAPACKE_ztprfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztprfs(armpl_int_t matrix_layout, char uplo, char trans,
                           char diag, armpl_int_t n, armpl_int_t nrhs,
                           const armpl_doublecomplex_t *ap,
                           const armpl_doublecomplex_t *b, armpl_int_t ldb,
                           const armpl_doublecomplex_t *x, armpl_int_t ldx,
                           double *ferr, double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctprfs*, *LAPACKE_dtprfs*, *LAPACKE_stprfs* and *LAPACKE_ztprfs*. It also exists with a native Fortran interface as *ztprfs*.

4.19.2367 LAPACKE_ztprfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztprfs_work(armpl_int_t matrix_layout, char uplo,
                                char trans, char diag, armpl_int_t n,
                                armpl_int_t nrhs,
                                const armpl_doublecomplex_t *ap,
                                const armpl_doublecomplex_t *b,
                                armpl_int_t ldb,
                                const armpl_doublecomplex_t *x,
                                armpl_int_t ldx, double *ferr, double *berr,
                                armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctprfs_work*, *LAPACKE_dtprfs_work*, *LAPACKE_stprfs_work* and *LAPACKE_ztprfs_work*.

4.19.2368 LAPACKE_ztptri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztptri(armpl_int_t matrix_layout, char uplo, char diag,
                           armpl_int_t n, armpl_doublecomplex_t *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctptri*, *LAPACKE_dtptri*, *LAPACKE_stptri* and *LAPACKE_ztptri*. It also exists with a native Fortran interface as *ztptri*.

4.19.2369 LAPACKE_ztptri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztptri_work(armpl_int_t matrix_layout, char uplo,
                                char diag, armpl_int_t n,
                                armpl_doublecomplex_t *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctptri_work*, *LAPACKE_dtptri_work*, *LAPACKE_stptri_work* and *LAPACKE_ztptri_work*.

4.19.2370 LAPACKE_ztptrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztptrs(armpl_int_t matrix_layout, char uplo, char trans,
                           char diag, armpl_int_t n, armpl_int_t nrhs,
                           const armpl_doublecomplex_t *ap,
                           armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctptrs*, *LAPACKE_dtptrs*, *LAPACKE_stptrs* and *LAPACKE_ztptrs*. It also exists with a native Fortran interface as *ztptrs*.

4.19.2371 LAPACKE_ztptrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztptrs_work(armpl_int_t matrix_layout, char uplo,
                                char trans, char diag, armpl_int_t n,
                                armpl_int_t nrhs,
                                const armpl_doublecomplex_t *ap,
                                armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctptrs_work*, *LAPACKE_dtptrs_work*, *LAPACKE_stptrs_work* and *LAPACKE_ztptrs_work*.

4.19.2372 LAPACKE_ztpttf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztpttf(armpl_int_t matrix_layout, char transr, char uplo,
                           armpl_int_t n, const armpl_doublecomplex_t *ap,
                           armpl_doublecomplex_t *arf);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpttf*, *LAPACKE_dtpttf*, *LAPACKE_stpttf* and *LAPACKE_ztpttf*. It also exists with a native Fortran interface as *ztpttf*.

4.19.2373 LAPACKE_ztpttf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztpttf_work(armpl_int_t matrix_layout, char transr,
                                char uplo, armpl_int_t n,
                                const armpl_doublecomplex_t *ap,
                                armpl_doublecomplex_t *arf);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpttf_work*, *LAPACKE_dtpttf_work*, *LAPACKE_stpttf_work* and *LAPACKE_ztpttf_work*.

4.19.2374 LAPACKE_ztpttr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztpttr(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, const armpl_doublecomplex_t *ap,
                           armpl_doublecomplex_t *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpttr*, *LAPACKE_dtpttr*, *LAPACKE_stpttr* and *LAPACKE_ztpttr*. It also exists with a native Fortran interface as *ztpttr*.

4.19.2375 LAPACKE_ztpttr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztpttr_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n,
                                const armpl_doublecomplex_t *ap,
                                armpl_doublecomplex_t *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctpttr_work*, *LAPACKE_dtpttr_work*, *LAPACKE_stpttr_work* and *LAPACKE_ztpttr_work*.

4.19.2376 LAPACKE_ztrcon

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztrcon(armpl_int_t matrix_layout, char norm, char uplo,
                           char diag, armpl_int_t n,
                           const armpl_doublecomplex_t *a, armpl_int_t lda,
                           double *rcond);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrcon*, *LAPACKE_dtrcon*, *LAPACKE_strcon* and *LAPACKE_ztrcon*. It also exists with a native Fortran interface as *ztrcon*.

4.19.2377 LAPACKE_ztrcon_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztrcon_work(armpl_int_t matrix_layout, char norm,
                                char uplo, char diag, armpl_int_t n,
                                const armpl_doublecomplex_t *a,
                                armpl_int_t lda, double *rcond,
                                armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrcon_work*, *LAPACKE_dtrcon_work*, *LAPACKE_strcon_work* and *LAPACKE_ztrcon_work*.

4.19.2378 LAPACKE_ztrevc

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztrevc(armpl_int_t matrix_layout, char side, char howmny,
                           const armpl_int_t *select, armpl_int_t n,
                           armpl_doublecomplex_t *t, armpl_int_t ldt,
                           armpl_doublecomplex_t *vl, armpl_int_t ldvl,
                           armpl_doublecomplex_t *vr, armpl_int_t ldvr,
                           armpl_int_t mm, armpl_int_t *m);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrevc*, *LAPACKE_dtrevc*, *LAPACKE_strevc* and *LAPACKE_ztrevc*. It also exists with a native Fortran interface as *ztrevc*.

4.19.2379 LAPACKE_ztrevc_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztrevc_work(armpl_int_t matrix_layout, char side,
                                char howmny, const armpl_int_t *select,
                                armpl_int_t n, armpl_doublecomplex_t *t,
                                armpl_int_t ldt, armpl_doublecomplex_t *vl,
                                armpl_int_t ldvl, armpl_doublecomplex_t *vr,
                                armpl_int_t ldvr, armpl_int_t mm,
                                armpl_int_t *m, armpl_doublecomplex_t *work,
                                double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrevc_work*, *LAPACKE_dtrevc_work*, *LAPACKE_strevc_work* and *LAPACKE_ztrevc_work*.

4.19.2380 LAPACKE_ztrexc

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztrexc(armpl_int_t matrix_layout, char compq,
                           armpl_int_t n, armpl_doublecomplex_t *t,
                           armpl_int_t ldt, armpl_doublecomplex_t *q,
                           armpl_int_t ldq, armpl_int_t ifst,
                           armpl_int_t ilst);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrexc*, *LAPACKE_dtrexc*, *LAPACKE_strexc* and *LAPACKE_ztrexc*. It also exists with a native Fortran interface as *ztrexc*.

4.19.2381 LAPACKE_ztrexc_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztrexc_work(armpl_int_t matrix_layout, char compq,
                                armpl_int_t n, armpl_doublecomplex_t *t,
                                armpl_int_t ldt, armpl_doublecomplex_t *q,
                                armpl_int_t ldq, armpl_int_t ifst,
                                armpl_int_t ilst);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrexc_work*, *LAPACKE_dtrexc_work*, *LAPACKE_strexc_work* and *LAPACKE_ztrexc_work*.

4.19.2382 LAPACKE_ztrrfs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztrrfs(armpl_int_t matrix_layout, char uplo, char trans,
                           char diag, armpl_int_t n, armpl_int_t nrhs,
                           const armpl_doublecomplex_t *a, armpl_int_t lda,
                           const armpl_doublecomplex_t *b, armpl_int_t ldb,
                           const armpl_doublecomplex_t *x, armpl_int_t ldx,
                           double *ferr, double *berr);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrrfs*, *LAPACKE_dtrrfs*, *LAPACKE_strrfs* and *LAPACKE_ztrrfs*. It also exists with a native Fortran interface as *ztrrfs*.

4.19.2383 LAPACKE_ztrrfs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztrrfs_work(armpl_int_t matrix_layout, char uplo,
                                char trans, char diag, armpl_int_t n,
                                armpl_int_t nrhs,
```

(continues on next page)

(continued from previous page)

```
                                const armpl_doublecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_doublecomplex_t *b,
                                armpl_int_t ldb,
                                const armpl_doublecomplex_t *x,
                                armpl_int_t ldx, double *ferr, double *berr,
                                armpl_doublecomplex_t *work, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrrfs_work*, *LAPACKE_dtrrfs_work*, *LAPACKE_strrfs_work* and *LAPACKE_ztrrfs_work*.

4.19.2384 LAPACKE_ztrsen

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztrsen(armpl_int_t matrix_layout, char job, char compq,
                           const armpl_int_t *select, armpl_int_t n,
                           armpl_doublecomplex_t *t, armpl_int_t ldt,
                           armpl_doublecomplex_t *q, armpl_int_t ldq,
                           armpl_doublecomplex_t *w, armpl_int_t *m,
                           double *s, double *sep);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrsen*, *LAPACKE_dtrsen*, *LAPACKE_strsen* and *LAPACKE_ztrsen*. It also exists with a native Fortran interface as *ztrsen*.

4.19.2385 LAPACKE_ztrsen_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztrsen_work(armpl_int_t matrix_layout, char job,
                                char compq, const armpl_int_t *select,
                                armpl_int_t n, armpl_doublecomplex_t *t,
                                armpl_int_t ldt, armpl_doublecomplex_t *q,
                                armpl_int_t ldq, armpl_doublecomplex_t *w,
                                armpl_int_t *m, double *s, double *sep,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrsen_work*, *LAPACKE_dtrsen_work*, *LAPACKE_strsen_work* and *LAPACKE_ztrsen_work*.

4.19.2386 LAPACKE_ztrsna

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztrsna(armpl_int_t matrix_layout, char job, char howmny,
                           const armpl_int_t *select, armpl_int_t n,
                           const armpl_doublecomplex_t *t, armpl_int_t ldt,
                           const armpl_doublecomplex_t *vl, armpl_int_t ldvl,
                           const armpl_doublecomplex_t *vr, armpl_int_t ldvr,
                           double *s, double *sep, armpl_int_t mm,
                           armpl_int_t *m);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrsna*, *LAPACKE_dtrsna*, *LAPACKE_strsna* and *LAPACKE_ztrsna*. It also exists with a native Fortran interface as *ztrsna*.

4.19.2387 LAPACKE_ztrsna_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztrsna_work(armpl_int_t matrix_layout, char job,
                                char howmny, const armpl_int_t *select,
                                armpl_int_t n, const armpl_doublecomplex_t *t,
                                armpl_int_t ldt,
                                const armpl_doublecomplex_t *vl,
                                armpl_int_t ldvl,
                                const armpl_doublecomplex_t *vr,
                                armpl_int_t ldvr, double *s, double *sep,
                                armpl_int_t mm, armpl_int_t *m,
                                armpl_doublecomplex_t *work,
                                armpl_int_t ldwork, double *rwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrsna_work*, *LAPACKE_dtrsna_work*, *LAPACKE_strsna_work* and *LAPACKE_ztrsna_work*.

4.19.2388 LAPACKE_ztrsyl

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztrsyl(armpl_int_t matrix_layout, char trana, char tranb,
                           armpl_int_t isgn, armpl_int_t m, armpl_int_t n,
                           const armpl_doublecomplex_t *a, armpl_int_t lda,
                           const armpl_doublecomplex_t *b, armpl_int_t ldb,
                           armpl_doublecomplex_t *c, armpl_int_t ldc,
                           double *scale);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrsyl*, *LAPACKE_dtrsyl*, *LAPACKE_strsyl* and *LAPACKE_ztrsyl*. It also exists with a native Fortran interface as *ztrsyl*.

4.19.2389 LAPACKE_ztrsyl_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztrsyl_work(armpl_int_t matrix_layout, char trana,
                                char tranb, armpl_int_t isgn, armpl_int_t m,
                                armpl_int_t n, const armpl_doublecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_doublecomplex_t *b,
                                armpl_int_t ldb, armpl_doublecomplex_t *c,
                                armpl_int_t ldc, double *scale);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrsyl_work*, *LAPACKE_dtrsyl_work*, *LAPACKE_strsyl_work* and *LAPACKE_ztrsyl_work*.

4.19.2390 LAPACKE_ztrtri

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztrtri(armpl_int_t matrix_layout, char uplo, char diag,
                           armpl_int_t n, armpl_doublecomplex_t *a,
                           armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrtri*, *LAPACKE_dtrtri*, *LAPACKE_strtri* and *LAPACKE_ztrtri*. It also exists with a native Fortran interface as *ztrtri*.

4.19.2391 LAPACKE_ztrtri_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztrtri_work(armpl_int_t matrix_layout, char uplo,
                                char diag, armpl_int_t n,
                                armpl_doublecomplex_t *a, armpl_int_t lda);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrtri_work*, *LAPACKE_dtrtri_work*, *LAPACKE_strtri_work* and *LAPACKE_ztrtri_work*.

4.19.2392 LAPACKE_ztrtrs

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztrtrs(armpl_int_t matrix_layout, char uplo, char trans,
                           char diag, armpl_int_t n, armpl_int_t nrhs,
                           const armpl_doublecomplex_t *a, armpl_int_t lda,
                           armpl_doublecomplex_t *b, armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrtrs*, *LAPACKE_dtrtrs*, *LAPACKE_strtrs* and *LAPACKE_ztrtrs*. It also exists with a native Fortran interface as *ztrtrs*.

4.19.2393 LAPACKE_ztrtrs_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztrtrs_work(armpl_int_t matrix_layout, char uplo,
                                char trans, char diag, armpl_int_t n,
                                armpl_int_t nrhs,
                                const armpl_doublecomplex_t *a,
                                armpl_int_t lda, armpl_doublecomplex_t *b,
                                armpl_int_t ldb);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrtrs_work*, *LAPACKE_dtrtrs_work*, *LAPACKE_strtrs_work* and *LAPACKE_ztrtrs_work*.

4.19.2394 LAPACKE_ztrttf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztrttf(armpl_int_t matrix_layout, char transr, char uplo,
                           armpl_int_t n, const armpl_doublecomplex_t *a,
                           armpl_int_t lda, armpl_doublecomplex_t *arf);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrttf*, *LAPACKE_dtrttf*, *LAPACKE_strttf* and *LAPACKE_ztrttf*. It also exists with a native Fortran interface as *ztrttf*.

4.19.2395 LAPACKE_ztrttf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztrttf_work(armpl_int_t matrix_layout, char transr,
                                char uplo, armpl_int_t n,
                                const armpl_doublecomplex_t *a,
                                armpl_int_t lda, armpl_doublecomplex_t *arf);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrttf_work*, *LAPACKE_dtrttf_work*, *LAPACKE_strttf_work* and *LAPACKE_ztrttf_work*.

4.19.2396 LAPACKE_ztrttp

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztrttp(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, const armpl_doublecomplex_t *a,
                           armpl_int_t lda, armpl_doublecomplex_t *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrttp*, *LAPACKE_dtrttp*, *LAPACKE_strttp* and *LAPACKE_ztrttp*. It also exists with a native Fortran interface as *ztrttp*.

4.19.2397 LAPACKE_ztrttp_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztrttp_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, const armpl_doublecomplex_t *a,
                                armpl_int_t lda, armpl_doublecomplex_t *ap);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctrttp_work*, *LAPACKE_dtrttp_work*, *LAPACKE_strttp_work* and *LAPACKE_ztrttp_work*.

4.19.2398 LAPACKE_ztzrzf

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztzrzf(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_doublecomplex_t *a,
                           armpl_int_t lda, armpl_doublecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctzrzf*, *LAPACKE_dtzrzf*, *LAPACKE_stzrzf* and *LAPACKE_ztzrzf*. It also exists with a native Fortran interface as *ztzrzf*.

4.19.2399 LAPACKE_ztzrzf_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_ztzrzf_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_doublecomplex_t *a,
                                armpl_int_t lda, armpl_doublecomplex_t *tau,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_ctzrzf_work*, *LAPACKE_dtzrzf_work*, *LAPACKE_stzrzf_work* and *LAPACKE_ztzrzf_work*.

4.19.2400 LAPACKE_zunbdb

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zunbdb(armpl_int_t matrix_layout, char trans, char signs,
                           armpl_int_t m, armpl_int_t p, armpl_int_t q,
                           armpl_doublecomplex_t *x11, armpl_int_t ldx11,
                           armpl_doublecomplex_t *x12, armpl_int_t ldx12,
```

(continues on next page)

(continued from previous page)

```
                                armpl_doublecomplex_t *x21, armpl_int_t ldx21,
                                armpl_doublecomplex_t *x22, armpl_int_t ldx22,
                                double *theta, double *phi,
                                armpl_doublecomplex_t *taup1,
                                armpl_doublecomplex_t *taup2,
                                armpl_doublecomplex_t *tauq1,
                                armpl_doublecomplex_t *tauq2);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunbdb* and *LAPACKE_zunbdb*. It also exists with a native Fortran interface as *zunbdb*.

4.19.2401 LAPACKE_zunbdb_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zunbdb_work(armpl_int_t matrix_layout, char trans,
                                char signs, armpl_int_t m, armpl_int_t p,
                                armpl_int_t q, armpl_doublecomplex_t *x11,
                                armpl_int_t ldx11, armpl_doublecomplex_t *x12,
                                armpl_int_t ldx12, armpl_doublecomplex_t *x21,
                                armpl_int_t ldx21, armpl_doublecomplex_t *x22,
                                armpl_int_t ldx22, double *theta, double *phi,
                                armpl_doublecomplex_t *taup1,
                                armpl_doublecomplex_t *taup2,
                                armpl_doublecomplex_t *tauq1,
                                armpl_doublecomplex_t *tauq2,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunbdb_work* and *LAPACKE_zunbdb_work*.

4.19.2402 LAPACKE_zuncsd

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zuncsd(armpl_int_t matrix_layout, char jobu1, char jobu2,
                           char jobv1t, char jobv2t, char trans, char signs,
                           armpl_int_t m, armpl_int_t p, armpl_int_t q,
                           armpl_doublecomplex_t *x11, armpl_int_t ldx11,
                           armpl_doublecomplex_t *x12, armpl_int_t ldx12,
                           armpl_doublecomplex_t *x21, armpl_int_t ldx21,
                           armpl_doublecomplex_t *x22, armpl_int_t ldx22,
                           double *theta, armpl_doublecomplex_t *u1,
                           armpl_int_t ldu1, armpl_doublecomplex_t *u2,
                           armpl_int_t ldu2, armpl_doublecomplex_t *v1t,
                           armpl_int_t ldv1t, armpl_doublecomplex_t *v2t,
                           armpl_int_t ldv2t);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cuncsd* and *LAPACKE_zuncsd*. It also exists with a native Fortran interface as *zuncsd*.

4.19.2403 LAPACKE_zuncsd2by1

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zuncsd2by1(armpl_int_t matrix_layout, char jobu1,
                               char jobu2, char jobv1t, armpl_int_t m,
                               armpl_int_t p, armpl_int_t q,
                               armpl_doublecomplex_t *x11, armpl_int_t ldx11,
                               armpl_doublecomplex_t *x21, armpl_int_t ldx21,
                               double *theta, armpl_doublecomplex_t *u1,
                               armpl_int_t ldu1, armpl_doublecomplex_t *u2,
                               armpl_int_t ldu2, armpl_doublecomplex_t *v1t,
                               armpl_int_t ldv1t);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cuncsd2by1* and *LAPACKE_zuncsd2by1*. It also exists with a native Fortran interface as *zuncsd2by1*.

4.19.2404 LAPACKE_zuncsd2by1_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zuncsd2by1_work(armpl_int_t matrix_layout, char jobu1,
                                    char jobu2, char jobv1t, armpl_int_t m,
                                    armpl_int_t p, armpl_int_t q,
                                    armpl_doublecomplex_t *x11,
                                    armpl_int_t ldx11,
                                    armpl_doublecomplex_t *x21,
                                    armpl_int_t ldx21, double *theta,
                                    armpl_doublecomplex_t *u1,
                                    armpl_int_t ldu1,
                                    armpl_doublecomplex_t *u2,
                                    armpl_int_t ldu2,
                                    armpl_doublecomplex_t *v1t,
                                    armpl_int_t ldv1t,
                                    armpl_doublecomplex_t *work,
                                    armpl_int_t lwork, double *rwork,
                                    armpl_int_t lrwork, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cuncsd2by1_work* and *LAPACKE_zuncsd2by1_work*.

4.19.2405 LAPACKE_zuncsd_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zuncsd_work(armpl_int_t matrix_layout, char jobu1,
                                char jobu2, char jobv1t, char jobv2t,
                                char trans, char signs, armpl_int_t m,
                                armpl_int_t p, armpl_int_t q,
                                armpl_doublecomplex_t *x11, armpl_int_t ldx11,
                                armpl_doublecomplex_t *x12, armpl_int_t ldx12,
                                armpl_doublecomplex_t *x21, armpl_int_t ldx21,
                                armpl_doublecomplex_t *x22, armpl_int_t ldx22,
                                double *theta, armpl_doublecomplex_t *u1,
                                armpl_int_t ldu1, armpl_doublecomplex_t *u2,
                                armpl_int_t ldu2, armpl_doublecomplex_t *v1t,
                                armpl_int_t ldv1t, armpl_doublecomplex_t *v2t,
                                armpl_int_t ldv2t,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork, double *rwork,
                                armpl_int_t lrwork, armpl_int_t *iwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cuncsd_work* and *LAPACKE_zuncsd_work*.

4.19.2406 LAPACKE_zungbr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zungbr(armpl_int_t matrix_layout, char vect,
                           armpl_int_t m, armpl_int_t n, armpl_int_t k,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           const armpl_doublecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cungbr* and *LAPACKE_zungbr*. It also exists with a native Fortran interface as *zungbr*.

4.19.2407 LAPACKE_zungbr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zungbr_work(armpl_int_t matrix_layout, char vect,
                                armpl_int_t m, armpl_int_t n, armpl_int_t k,
                                armpl_doublecomplex_t *a, armpl_int_t lda,
                                const armpl_doublecomplex_t *tau,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cungbr_work* and *LAPACKE_zungbr_work*.

4.19.2408 LAPACKE_zunghr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zunghr(armpl_int_t matrix_layout, armpl_int_t n,
                           armpl_int_t ilo, armpl_int_t ihi,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           const armpl_doublecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.

- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunghr* and *LAPACKE_zunghr*. It also exists with a native Fortran interface as *zunghr*.

4.19.2409 LAPACKE_zunghr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zunghr_work(armpl_int_t matrix_layout, armpl_int_t n,
                                armpl_int_t ilo, armpl_int_t ihi,
                                armpl_doublecomplex_t *a, armpl_int_t lda,
                                const armpl_doublecomplex_t *tau,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunghr_work* and *LAPACKE_zunghr_work*.

4.19.2410 LAPACKE_zunglq

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zunglq(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t k,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           const armpl_doublecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunglq* and *LAPACKE_zunglq*. It also exists with a native Fortran interface as *zunglq*.

4.19.2411 LAPACKE_zunglq_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zunglq_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t k,
                                armpl_doublecomplex_t *a, armpl_int_t lda,
                                const armpl_doublecomplex_t *tau,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunglq_work* and *LAPACKE_zunglq_work*.

4.19.2412 LAPACKE_zungql

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zungql(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t k,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           const armpl_doublecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cungql* and *LAPACKE_zungql*. It also exists with a native Fortran interface as *zungql*.

4.19.2413 LAPACKE_zungql_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zungql_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t k,
                                armpl_doublecomplex_t *a, armpl_int_t lda,
                                const armpl_doublecomplex_t *tau,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cungql_work* and *LAPACKE_zungql_work*.

4.19.2414 LAPACKE_zungqr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zungqr(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t k,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           const armpl_doublecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cungqr* and *LAPACKE_zungqr*. It also exists with a native Fortran interface as *zungqr*.

4.19.2415 LAPACKE_zungqr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zungqr_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t k,
                                armpl_doublecomplex_t *a, armpl_int_t lda,
                                const armpl_doublecomplex_t *tau,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cungqr_work* and *LAPACKE_zungqr_work*.

4.19.2416 LAPACKE_zungrq

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zungrq(armpl_int_t matrix_layout, armpl_int_t m,
                           armpl_int_t n, armpl_int_t k,
                           armpl_doublecomplex_t *a, armpl_int_t lda,
                           const armpl_doublecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cungrq* and *LAPACKE_zungrq*. It also exists with a native Fortran interface as *zungrq*.

4.19.2417 LAPACKE_zungrq_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zungrq_work(armpl_int_t matrix_layout, armpl_int_t m,
                                armpl_int_t n, armpl_int_t k,
                                armpl_doublecomplex_t *a, armpl_int_t lda,
                                const armpl_doublecomplex_t *tau,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cungrq_work* and *LAPACKE_zungrq_work*.

4.19.2418 LAPACKE_zungtr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zungtr(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, armpl_doublecomplex_t *a,
                           armpl_int_t lda,
                           const armpl_doublecomplex_t *tau);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cungtr* and *LAPACKE_zungtr*. It also exists with a native Fortran interface as *zungtr*.

4.19.2419 LAPACKE_zungtr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zungtr_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n, armpl_doublecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_doublecomplex_t *tau,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cungtr_work* and *LAPACKE_zungtr_work*.

4.19.2420 LAPACKE_zunmbr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zunmbr(armpl_int_t matrix_layout, char vect, char side,
                           char trans, armpl_int_t m, armpl_int_t n,
                           armpl_int_t k, const armpl_doublecomplex_t *a,
                           armpl_int_t lda, const armpl_doublecomplex_t *tau,
                           armpl_doublecomplex_t *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunmbr* and *LAPACKE_zunmbr*. It also exists with a native Fortran interface as *zunmbr*.

4.19.2421 LAPACKE_zunmbr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zunmbr_work(armpl_int_t matrix_layout, char vect,
                                char side, char trans, armpl_int_t m,
                                armpl_int_t n, armpl_int_t k,
                                const armpl_doublecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_doublecomplex_t *tau,
                                armpl_doublecomplex_t *c, armpl_int_t ldc,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunmbr_work* and *LAPACKE_zunmbr_work*.

4.19.2422 LAPACKE_zunmhr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zunmhr(armpl_int_t matrix_layout, char side, char trans,
                           armpl_int_t m, armpl_int_t n, armpl_int_t ilo,
                           armpl_int_t ihi, const armpl_doublecomplex_t *a,
                           armpl_int_t lda, const armpl_doublecomplex_t *tau,
                           armpl_doublecomplex_t *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunmhr* and *LAPACKE_zunmhr*. It also exists with a native Fortran interface as *zunmhr*.

4.19.2423 LAPACKE_zunmhr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zunmhr_work(armpl_int_t matrix_layout, char side,
                                char trans, armpl_int_t m, armpl_int_t n,
                                armpl_int_t ilo, armpl_int_t ihi,
                                const armpl_doublecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_doublecomplex_t *tau,
                                armpl_doublecomplex_t *c, armpl_int_t ldc,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunmhr_work* and *LAPACKE_zunmhr_work*.

4.19.2424 LAPACKE_zunmlq

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zunmlq(armpl_int_t matrix_layout, char side, char trans,
                           armpl_int_t m, armpl_int_t n, armpl_int_t k,
                           const armpl_doublecomplex_t *a, armpl_int_t lda,
                           const armpl_doublecomplex_t *tau,
                           armpl_doublecomplex_t *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunmlq* and *LAPACKE_zunmlq*. It also exists with a native Fortran interface as *zunmlq*.

4.19.2425 LAPACKE_zunmlq_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zunmlq_work(armpl_int_t matrix_layout, char side,
                                char trans, armpl_int_t m, armpl_int_t n,
                                armpl_int_t k, const armpl_doublecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_doublecomplex_t *tau,
                                armpl_doublecomplex_t *c, armpl_int_t ldc,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunmlq_work* and *LAPACKE_zunmlq_work*.

4.19.2426 LAPACKE_zunmql

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zunmql(armpl_int_t matrix_layout, char side, char trans,
                           armpl_int_t m, armpl_int_t n, armpl_int_t k,
                           const armpl_doublecomplex_t *a, armpl_int_t lda,
                           const armpl_doublecomplex_t *tau,
                           armpl_doublecomplex_t *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunmql* and *LAPACKE_zunmql*. It also exists with a native Fortran interface as *zunmql*.

4.19.2427 LAPACKE_zunmql_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zunmql_work(armpl_int_t matrix_layout, char side,
                                char trans, armpl_int_t m, armpl_int_t n,
                                armpl_int_t k, const armpl_doublecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_doublecomplex_t *tau,
                                armpl_doublecomplex_t *c, armpl_int_t ldc,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer

- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunmql_work* and *LAPACKE_zunmql_work*.

4.19.2428 LAPACKE_zunmqr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zunmqr(armpl_int_t matrix_layout, char side, char trans,
                           armpl_int_t m, armpl_int_t n, armpl_int_t k,
                           const armpl_doublecomplex_t *a, armpl_int_t lda,
                           const armpl_doublecomplex_t *tau,
                           armpl_doublecomplex_t *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunmqr* and *LAPACKE_zunmqr*. It also exists with a native Fortran interface as *zunmqr*.

4.19.2429 LAPACKE_zunmqr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zunmqr_work(armpl_int_t matrix_layout, char side,
                                char trans, armpl_int_t m, armpl_int_t n,
                                armpl_int_t k, const armpl_doublecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_doublecomplex_t *tau,
                                armpl_doublecomplex_t *c, armpl_int_t ldc,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunmqr_work* and *LAPACKE_zunmqr_work*.

4.19.2430 LAPACKE_zunmrq

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zunmrq(armpl_int_t matrix_layout, char side, char trans,
                           armpl_int_t m, armpl_int_t n, armpl_int_t k,
                           const armpl_doublecomplex_t *a, armpl_int_t lda,
                           const armpl_doublecomplex_t *tau,
                           armpl_doublecomplex_t *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunmrq* and *LAPACKE_zunmrq*. It also exists with a native Fortran interface as *zunmrq*.

4.19.2431 LAPACKE_zunmrq_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zunmrq_work(armpl_int_t matrix_layout, char side,
                                char trans, armpl_int_t m, armpl_int_t n,
                                armpl_int_t k, const armpl_doublecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_doublecomplex_t *tau,
```

(continues on next page)

(continued from previous page)

```
                                armpl_doublecomplex_t *c, armpl_int_t ldc,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunmrq_work* and *LAPACKE_zunmrq_work*.

4.19.2432 LAPACKE_zunmrz

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zunmrz(armpl_int_t matrix_layout, char side, char trans,
                           armpl_int_t m, armpl_int_t n, armpl_int_t k,
                           armpl_int_t l, const armpl_doublecomplex_t *a,
                           armpl_int_t lda, const armpl_doublecomplex_t *tau,
                           armpl_doublecomplex_t *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunmrz* and *LAPACKE_zunmrz*. It also exists with a native Fortran interface as *zunmrz*.

4.19.2433 LAPACKE_zunmrz_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zunmrz_work(armpl_int_t matrix_layout, char side,
                                char trans, armpl_int_t m, armpl_int_t n,
                                armpl_int_t k, armpl_int_t l,
                                const armpl_doublecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_doublecomplex_t *tau,
                                armpl_doublecomplex_t *c, armpl_int_t ldc,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunmrz_work* and *LAPACKE_zunmrz_work*.

4.19.2434 LAPACKE_zunmtr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zunmtr(armpl_int_t matrix_layout, char side, char uplo,
                           char trans, armpl_int_t m, armpl_int_t n,
                           const armpl_doublecomplex_t *a, armpl_int_t lda,
                           const armpl_doublecomplex_t *tau,
                           armpl_doublecomplex_t *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunmtr* and *LAPACKE_zunmtr*. It also exists with a native Fortran interface as *zunmtr*.

4.19.2435 LAPACKE_zunmtr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zunmtr_work(armpl_int_t matrix_layout, char side,
                                char uplo, char trans, armpl_int_t m,
                                armpl_int_t n, const armpl_doublecomplex_t *a,
                                armpl_int_t lda,
                                const armpl_doublecomplex_t *tau,
                                armpl_doublecomplex_t *c, armpl_int_t ldc,
                                armpl_doublecomplex_t *work,
                                armpl_int_t lwork);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cunmtr_work* and *LAPACKE_zunmtr_work*.

4.19.2436 LAPACKE_zupgtr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zupgtr(armpl_int_t matrix_layout, char uplo,
                           armpl_int_t n, const armpl_doublecomplex_t *ap,
                           const armpl_doublecomplex_t *tau,
                           armpl_doublecomplex_t *q, armpl_int_t ldq);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cupgtr* and *LAPACKE_zupgtr*. It also exists with a native Fortran interface as *zupgtr*.

4.19.2437 LAPACKE_zupgtr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zupgtr_work(armpl_int_t matrix_layout, char uplo,
                                armpl_int_t n,
                                const armpl_doublecomplex_t *ap,
                                const armpl_doublecomplex_t *tau,
                                armpl_doublecomplex_t *q, armpl_int_t ldq,
                                armpl_doublecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cupgtr_work* and *LAPACKE_zupgtr_work*.

4.19.2438 LAPACKE_zupmtr

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zupmtr(armpl_int_t matrix_layout, char side, char uplo,
                           char trans, armpl_int_t m, armpl_int_t n,
                           const armpl_doublecomplex_t *ap,
                           const armpl_doublecomplex_t *tau,
                           armpl_doublecomplex_t *c, armpl_int_t ldc);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cupmtr* and *LAPACKE_zupmtr*. It also exists with a native Fortran interface as *zupmtr*.

4.19.2439 LAPACKE_zupmtr_work

Syntax

```
#include "armpl.h"

armpl_int_t LAPACKE_zupmtr_work(armpl_int_t matrix_layout, char side,
                                char uplo, char trans, armpl_int_t m,
                                armpl_int_t n,
                                const armpl_doublecomplex_t *ap,
                                const armpl_doublecomplex_t *tau,
                                armpl_doublecomplex_t *c, armpl_int_t ldc,
                                armpl_doublecomplex_t *work);
```

Parameters

LAPACKE routines operate similarly to the equivalent Fortran routine with minor changes.

- All scalar quantities passed in by value.
- All arrays are passed in by pointer
- The first parameter is usually `matrix_layout`. This takes a value of either `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, depending on whether the data is laid out in memory in row-major or column-major format, respectively.

Related Information

For this routine in other precisions, please see *LAPACKE_cupmtr_work* and *LAPACKE_zupmtr_work*.

CHAPTER

FIVE

FAST FOURIER TRANSFORMS FFTS

5.1 Fast Fourier Transforms FFTs Introduction

For Fast Fourier Transforms Arm Performance Libraries uses the same interface as FFTW3. In addition to the material provided here, users are also referred to the FFTW3 Website for additional documentation:

http://www.fftw.org/fftw3_doc/

FFTW3 support in Arm Performance Libraries currently covers, in C and Fortran, and in single and double precision, the basic, advanced and guru functions with `_dft_` in the function name. In addition, half precision interfaces are provided in C. Half precision support is an extension to the FFTW3 interface which uses fftwh as the prefix for relevant functions and types.

Existing source code calling in to the supported FFTW3 functions need not be modified. Include `fftw3.h` as normal in your source code. Both the single and double precision functions are to be found in one library.

Where plans are not currently supported, i.e. the real-to-real transforms, the interface will return a NULL pointer except for MPI functions. This matches the listed behaviour of FFTW.

Note that since the basic and advanced FFTW interfaces explicitly specify `int` and `unsigned` datatypes then even for ilp64 Arm Performance Libraries the supported FFTW interfaces support only 32-bit integers. For transforms requiring 64-bit integers users are referred to the guru interface.

Behaviour is based on FFTW3 at version 3.3.8.

5.2 How to choose which FFT routine you need

The FFT routines provided in Arm Performance Libraries are grouped into the basic, advanced and guru interfaces. In addition, like all other routines in Arm Performance Libraries they are available in single and double precision and are available in both C and Fortran.

We recommend using the most complicated single call you need to in order to completely describe the set of FFT computations you need to perform, but no more complicated. For instance if the advanced interface covers everything you need, do not use the guru interface. Within Arm Performance Libraries you will be utilising the same high performing algorithms. The basic interface is a small subset of the advanced interface, and the guru interface allows an even greater degree of flexibility over the data layout.

In order to choose the correct routine to use it is useful to understand the different categories:

- Complex-to-complex routines can be applied in both forward and backward directions
- Real-to-complex routines transform real input into complex Hermitian output
- Complex-to-real routines transform from complex Hermitian input to real output

For this discussion we will stick to complex-to-complex functions, although exactly the same rules, with a similar nomenclature, applies for real transforms. Note also that whilst the descriptions talk about “solving FFT problems” the distinction between the interface is all made at the planning stage, with the created plan being the object that is ultimately executed.

5.2.1 The basic interface

In its simplest form the *basic interface* is ideal for cases where a single FFT needs to be solved and the data is all contiguous in memory. This has flavours depending on the number of dimensions, namely *1*-d, *2*-d, *3*-d or *n*-d. The respective plans are *fftw_plan_dft_1d*, *fftw_plan_dft_2d*, *fftw_plan_dft_3d*, and *fftw_plan_dft*.

The *n*-dimensional plan provides us our first layer of abstraction of the interface. For example, plans `p1` and `p2` for a 2-d transform of size 3x4 would be equivalent when made using the following two calls:

C specification:

```
/* Basic interface - 2-d */
p1 = fftw_plan_dft_2d(3, 4, in, out, sign, flags);

/* Basic interface - n-d */
n[0] = 3; n[1] = 4;
p2 = fftw_plan_dft(2, n, in, out, sign, flags);
```

Obviously the n-dimensional interface allows problems with an arbitrary number of dimensions to be solved.

5.2.2 The advanced interface

The *advanced interface* is to be used whenever there are multiple identically sized FFTs to be solved that are laid out regularly in memory, although not necessarily contiguously. The three main options that can be set are:

- Specifying multiple inputs to be solved. In this case, rather than having a loop around calling the basic interface `HOWMANY` times, where the memory distance between input and output cases are given by `IDIST` and `ODIST`, creating a separate plan for each case, you can have a single call to *fftw_plan_many_dft* that creates a single plan.

  This is shown by comparing the following two cases which produce FFT plans on the same data.

  ```
  /* Basic interface */
  for (i=0; i<HOWMANY; i++)
  {
      p3[i] = fftw_plan_dft_2d(3, 4, in[i*IDIST], out[i*ODIST], sign, flags);
  }

  /* Advanced interface */
  p4 = fftw_plan_many_dft(2, n, HOWMANY, in, NULL, 1, IDIST, out, NULL, 1, ODIST,
  ↪ sign, flags);
  ```

- Inputs (and outputs) allow the possibility of 'striding' through memory, meaning that consecutive elements along a dimension might not be adjacent in memory. This is a situation that cannot be handled through the basic interface.

  In the example above the distance between successive elements is 1 for both input (`in`) and output (`out`) arrays, and this is specified as input arguments 6 and 10 respectively. These need not be the same as each other, dependent on your use case.

- The final generalisation allowed in the advanced interface is that the dimensions of each FFT being solved are actually subarrays of a larger memory space. This means that the length of each dimension, `n[i]`, might actually be embedded in a larger memory space. These are given through the parameters `inembed` and `onembed` which define a memory space of `rank` dimensions.

  If the problem to be solved is such that the `inembed[i]==n[i]` for all `i`, then the `inembed` and `onembed` parameters can be passed in as as `NULL`.

5.2.3 The guru interface

The generalisations possible through the advanced interface are further extended using the *guru interface*. The guru interface allows:

1. different strides between data in each dimension
2. the generalisation of the `howmany` data to now be a multidimensional structure
3. the splitting of real and imaginary parts.

Calls to the guru interface introduce the use of an array of structures of type `fftw_iodim`. The precise details of the contents of this structure are explained on the relevant function pages, so are just described at a higher level here.

- The `fftdims` array of structures in *fftw_plan_guru_dft* is of length `rank`, and gives information about the structure of the data stored in that dimension. The three elements stored are

  `n`: the length of the FFT to be performed in that dimension,

  `is`: the stride between successive elements of the input array, and

  `os`: the stride between successive elements of the output array.

- The concept of a multidimensional `howmany` structure is for situations where you are solving multiple FFT transforms where the memory layout is not necessarily in a regular 1-d line, such as with the advanced interface, but you can imagine the start addresses being mapped onto a multi-dimensional grid.

  For example, if you have five sets of four complex-to-complex FFTs to solve, where:

  - each are 3x4 like the example above,
  - the `idist` in the first dimension was 256,
  - the gap between the start of each set of four was 4096,

  then this could be specified as:

```
/* Guru interface for multi-dimensional howmany */

rank = 2;
dims[0].n = 3;    dims[0].is = 4;    dims[0].os = 4;
dims[1].n = 4;    dims[1].is = 1;    dims[1].os = 1;

howmany_rank = 2;
howmany_dims[0].n = 4;    howmany_dims[0].is =  256;    howmany_dims[0].os = ␣
↪256;
howmany_dims[1].n = 5;    howmany_dims[1].is = 4096;    howmany_dims[1].os =␣
↪4096;

p5 = fftw_plan_guru_dft(rank, dims, howmany_rank, howmany_dims, in, out, sign,␣
↪flags);
```

- The splitting of real and imaginary parts mean that if your application has real and imaginary parts of the complex numbers stored separately then their locations can be passed in without having to separately copy them into an interleaved buffer. This is not a situation that can map onto the examples above, however is relatively straightforward to visualize, simply through the addition of two extra arguments passed to the planning call, pointers to the real and imaginary components of the input and output arrays replacing the existing pointers to `in` and `out`.

A final quirk of the guru interface is that if `howmany_dims` is given as 0 then a single FFT will be performed.

5.3 FFT complex to complex functions

5.3.1 fftw_plan_dft

`fftw_plan_dft` is a basic FFTW interface call for planning an *n*-dimensional complex-to-complex FFT operation on a single, contiguous data sequence.

Note: `in` and `out` can be the same pointer for an "in-place" transform.

Syntax

C specification:

```
#include "fftw3.h"

fftw_plan fftw_plan_dft(int rank, const int *n, fftw_complex *in,
                        fftw_complex *out, int sign, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_plan_dft(rank,n,in,out,sign,flags) &
    bind(C, name='fftw_plan_dft')
  integer(C_INT), value :: rank
  integer(C_INT), dimension(*), intent(in) :: n
  complex(C_DOUBLE_COMPLEX), dimension(*), intent(out) :: in
  complex(C_DOUBLE_COMPLEX), dimension(*), intent(out) :: out
  integer(C_INT), value :: sign
  integer(C_INT), value :: flags
end function fftw_plan_dft
```

Returns

An `fftw_plan` object is returned that can be used in subsequent *fftw_execute*`()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftw_execute_dft*`()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of the FFT problem to be solved

Constraint: `rank` >= 1

`n` Input parameter

`n` is `const int*`.

`n` is an array of dimension `[rank]`.

`n` contains the size of each dimension of the FFT to be solved

Constraint: `n[i]` >= 1 for i in 0, `rank-1`

`in` Input parameter

`in` is `fftw_complex*`

`in` is a pointer to a previously allocated array of `fftw_complex` data. At `fftw_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

out Input parameter

out is fftw_complex*

out is a pointer to a previously allocated array of fftw_complex data. At fftw_execute() time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

sign Input parameter

sign is int

sign is an integer denoting whether a forward or backward FFT transform should be performed

Constraint: sign = -1 (FFTW_FORWARD) for a forward transform,

sign = 1 (FFTW_BACKWARD) for a backwards transform

flags Input parameter

flags is unsigned

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in flags are:

- FFTW_ESTIMATE – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- FFTW_MEASURE – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- FFTW_PATIENT – Produce a plan, testing more options than FFTW_MEASURE. Input and output data might be overwritten.
- FFTW_EXHAUSTIVE – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- FFTW_WISDOM_ONLY – Produce a plan only if the requested case has existing wisdom, otherwise return NULL.
- FFTW_PRESERVE_INPUT – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- FFTW_DESTROY_INPUT – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- FFTW_CONSERVE_MEMORY – Request that extra memory is not created. This might be ignored.
- FFTW_UNALIGNED – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by FFTW_MEASURE, FFTW_PATIENT and FFTW_EXHAUSTIVE will be limited when using *fftw_set_timelimit*.
- If multiple options for planning rigor are included in flags, then the highest level of planning requested will be used.

5.3.2 fftw_plan_dft_1d

fftw_plan_dft_1d is a basic FFTW interface call for planning a 1-d complex-to-complex FFT operation on a single, contiguous data sequence.

Note: in and out can be the same pointer for an "in-place" transform.

Syntax

C specification:

```
#include "fftw3.h"

fftw_plan fftw_plan_dft_1d(int n0, fftw_complex *in, fftw_complex *out,
                           int sign, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_plan_dft_1d(n,in,out,sign,flags) &
    bind(C, name='fftw_plan_dft_1d')
  integer(C_INT), value :: n
  complex(C_DOUBLE_COMPLEX), dimension(*), intent(out) :: in
  complex(C_DOUBLE_COMPLEX), dimension(*), intent(out) :: out
  integer(C_INT), value :: sign
  integer(C_INT), value :: flags
end function fftw_plan_dft_1d
```

Returns

An `fftw_plan` object is returned that can be used in subsequent *fftw_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftw_execute_dft* `()`.

Parameters

`n0` Input parameter

`n0` is `int`

This is the length of the FFT sequence

Constraint: `n0` >= 1

`in` Input parameter

`in` is `fftw_complex*`

`in` is a pointer to a previously allocated array of `fftw_complex` data. At `fftw_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `fftw_complex*`

`out` is a pointer to a previously allocated array of `fftw_complex` data. At `fftw_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`sign` Input parameter

`sign` is `int`

`sign` is an integer denoting whether a forward or backward FFT transform should be performed

Constraint: `sign` = -1 (`FFTW_FORWARD`) for a forward transform,

`sign` = 1 (`FFTW_BACKWARD`) for a backwards transform

flags Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftw_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.3.3 fftw_plan_dft_2d

`fftw_plan_dft_2d` is a basic FFTW interface call for planning a 2-d complex-to-complex FFT operation on a single, contiguous data sequence.

Note: `in` and `out` can be the same pointer for an "in-place" transform.

Syntax

C specification:

```
#include "fftw3.h"

fftw_plan fftw_plan_dft_2d(int n0, int n1, fftw_complex *in,
                           fftw_complex *out, int sign, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_plan_dft_2d(n0,n1,in,out,sign,flags) &
    bind(C, name='fftw_plan_dft_2d')
  integer(C_INT), value :: n0
  integer(C_INT), value :: n1
  complex(C_DOUBLE_COMPLEX), dimension(*), intent(out) :: in
  complex(C_DOUBLE_COMPLEX), dimension(*), intent(out) :: out
  integer(C_INT), value :: sign
  integer(C_INT), value :: flags
end function fftw_plan_dft_2d
```

Returns

An `fftw_plan` object is returned that can be used in subsequent *fftw_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftw_execute_dft* `()`.

Parameters

`n0` Input parameter

`n0` is `int`

This is the length of the first dimension of the FFT sequence

Constraint: `n0` >= 1

`n1` Input parameter

`n1` is `int`

This is the length of the second dimension of the FFT sequence

Constraint: `n1` >= 1

`in` Input parameter

`in` is `fftw_complex*`

`in` is a pointer to a previously allocated array of `fftw_complex` data. At `fftw_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

out is fftw_complex

out is a pointer to a previously allocated array of fftw_complex data. At fftw_execute() time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`sign` Input parameter

`sign` is `int`

`sign` is an integer denoting whether a forward or backward FFT transform should be performed

Constraint: `sign` = -1 (`FFTW_FORWARD`) for a forward transform,

`sign` = 1 (`FFTW_BACKWARD`) for a backwards transform

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftw_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.3.4 fftw_plan_dft_3d

`fftw_plan_dft_3d` is a basic FFTW interface call for planning a 3-d complex-to-complex FFT operation on a single, contiguous data sequence.

Note: `in` and `out` can be the same pointer for an "in-place" transform.

Syntax

C specification:

```
#include "fftw3.h"

fftw_plan fftw_plan_dft_3d(int n0, int n1, int n2, fftw_complex *in,
                           fftw_complex *out, int sign, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_plan_dft_3d(n0,n1,n2,in,out,sign,flags) &
    bind(C, name='fftw_plan_dft_3d')
```

(continues on next page)

(continued from previous page)

```
  integer(C_INT), value :: n0
  integer(C_INT), value :: n1
  integer(C_INT), value :: n2
  complex(C_DOUBLE_COMPLEX), dimension(*), intent(out) :: in
  complex(C_DOUBLE_COMPLEX), dimension(*), intent(out) :: out
  integer(C_INT), value :: sign
  integer(C_INT), value :: flags
end function fftw_plan_dft_3d
```

Returns

An `fftw_plan` object is returned that can be used in subsequent *fftw_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftw_execute_dft* `()`.

Parameters

`n0` Input parameter

`n0` is `int`

This is the length of the first dimension of the FFT sequence

Constraint: `n0` >= 1

`n1` Input parameter

`n1` is `int`

This is the length of the second dimension of the FFT sequence

Constraint: `n1` >= 1

`n2` Input parameter

`n2` is `int`

This is the length of the third dimension of the FFT sequence

Constraint: `n2` >= 1

`in` Input parameter

`in` is `fftw_complex*`

`in` is a pointer to a previously allocated array of `fftw_complex` data. At `fftw_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `fftw_complex*`

`out` is a pointer to a previously allocated array of `fftw_complex` data. At `fftw_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`sign` Input parameter

`sign` is `int`

`sign` is an integer denoting whether a forward or backward FFT transform should be performed

Constraint: `sign` = -1 (`FFTW_FORWARD`) for a forward transform,

`sign` = 1 (`FFTW_BACKWARD`) for a backwards transform

flags Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftw_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.3.5 fftw_plan_guru_dft

`fftw_plan_guru_dft` is a guru FFTW interface call for planning an *n*-dimensional complex-to-complex FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner.

Syntax

C specification:

```
#include "fftw3.h"

fftw_plan fftw_plan_guru_dft(int rank, const fftw_iodim *dims,
                             int howmany_rank,
                             const fftw_iodim *howmany_dims,
                             fftw_complex *in, fftw_complex *out,
                             int sign, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_plan_guru_dft(rank,dims,howmany_rank, &
    howmany_dims,in,out,sign,flags) &
    bind(C, name='fftw_plan_guru_dft')
  integer(C_INT), value :: rank
  type(fftw_iodim), dimension(*), intent(in) :: dims
  integer(C_INT), value :: howmany_rank
  type(fftw_iodim), dimension(*), intent(in) :: howmany_dims
  complex(C_DOUBLE_COMPLEX), dimension(*), intent(out) :: in
  complex(C_DOUBLE_COMPLEX), dimension(*), intent(out) :: out
  integer(C_INT), value :: sign
  integer(C_INT), value :: flags
end function fftw_plan_guru_dft
```

Returns

An `fftw_plan` object is returned that can be used in subsequent *fftw_execute*`()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftw_execute_dft*`()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

`dims` Input parameter

`dims` is `const fftw_iodim *`

`dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the length of the `i` th dimension.
- `int is` – the stride between successive elements in the input vector in the `i` th dimension.
- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: `dims.n[i]` >= 1 for i in 0, `rank-1`

`howmany_rank` Input parameter

`howmany_rank` is `int`

`howmany_rank` is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then `howmany_rank` needs only be 1.

Constraint: `howmany` >= 0

Note: If `howmany_rank=0`, then a single FFT transform is computed.

`howmany_dims` Input parameter

`howmany_dims` is `const fftw_iodim *`

howmany_dims is an array of structures, dimension [rank], where the members of dims[i], for i in 0, rank-1, are defined to be:

- int n – the number of transforms to be computed of the i th dimension.of the howmany_rank-dimensional space mapped by the transforms to be computed.
- int is – the number of memory elements, of type fftw_complex, between the start of transforms in the i th dimension. of the howmany_rank-dimensional space mapped by the transforms to be computed.
- int os – the stride between successive elements in the output vector in the i th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the howmany_dims.n[] entries. For example, if howmany_rank=1, then howmany_dims[0].n is the memory distance between successive transforms.

Constraint: howmany_dims.n[i] >= 1 for i in 0, howmany_rank-1

in Input parameter

in is fftw_complex*

in is a pointer to a previously allocated array of fftw_complex data. At fftw_execute() time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

out Input parameter

out is fftw_complex*

out is a pointer to a previously allocated array of fftw_complex data. At fftw_execute() time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

sign Input parameter

sign is int

sign is an integer denoting whether a forward or backward FFT transform should be performed

Constraint: sign = -1 (FFTW_FORWARD) for a forward transform,

sign = 1 (FFTW_BACKWARD) for a backwards transform

flags Input parameter

flags is unsigned

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in flags are:

- FFTW_ESTIMATE – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- FFTW_MEASURE – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- FFTW_PATIENT – Produce a plan, testing more options than FFTW_MEASURE. Input and output data might be overwritten.
- FFTW_EXHAUSTIVE – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- FFTW_WISDOM_ONLY – Produce a plan only if the requested case has existing wisdom, otherwise return NULL.

- FFTW_PRESERVE_INPUT – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- FFTW_DESTROY_INPUT – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- FFTW_CONSERVE_MEMORY – Request that extra memory is not created. This might be ignored.
- FFTW_UNALIGNED – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by FFTW_MEASURE, FFTW_PATIENT and FFTW_EXHAUSTIVE will be limited when using *fftw_set_timelimit*.
- If multiple options for planning rigor are included in flags, then the highest level of planning requested will be used.

5.3.6 fftw_plan_guru_split_dft

fftw_plan_guru_split_dft is a guru FFTW interface call for planning an *n*-dimensional complex-to-complex FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner. Input and output data is stored in separate arrays for real and imaginary parts.

Syntax

C specification:

```
#include "fftw3.h"

fftw_plan fftw_plan_guru_split_dft(int rank, const fftw_iodim *dims,
                                   int howmany_rank,
                                   const fftw_iodim *howmany_dims,
                                   double *ri, double *ii, double *ro,
                                   double *io, int sign, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_plan_guru_split_dft(rank,dims,howmany_rank, &
    howmany_dims,ri,ii,ro,io,flags) &
    bind(C, name='fftw_plan_guru_split_dft')
  integer(C_INT), value :: rank
  type(fftw_iodim), dimension(*), intent(in) :: dims
  integer(C_INT), value :: howmany_rank
  type(fftw_iodim), dimension(*), intent(in) :: howmany_dims
  real(C_DOUBLE), dimension(*), intent(out) :: ri
  real(C_DOUBLE), dimension(*), intent(out) :: ii
  real(C_DOUBLE), dimension(*), intent(out) :: ro
  real(C_DOUBLE), dimension(*), intent(out) :: io
  integer(C_INT), value :: flags
end function fftw_plan_guru_split_dft
```

Returns

An fftw_plan object is returned that can be used in subsequent *fftw_execute*() calls on the previously supplied in and out arrays, or using different arrays via *fftw_execute_dft*().

Parameters

rank Input parameter

rank is int

The number of dimensions of each individual FFT problem to be solved.

Constraint: rank >= 1

dims Input parameter

dims is const fftw_iodim *

dims is an array of structures, dimension [rank], where the members of dims[i], for i in 0, rank-1, are defined to be:

- int n – the length of the i th dimension.
- int is – the stride between successive elements in the input vector in the i th dimension.
- int os – the stride between successive elements in the output vector in the i th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: dims.n[i] >= 1 for i in 0, rank-1

howmany_rank Input parameter

howmany_rank is int

howmany_rank is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then howmany_rank needs only be 1.

Constraint: howmany >= 0

Note: If howmany_rank=0, then a single FFT transform is computed.

howmany_dims Input parameter

howmany_dims is const fftw_iodim *

howmany_dims is an array of structures, dimension [rank], where the members of dims[i], for i in 0, rank-1, are defined to be:

- int n – the number of transforms to be computed of the i th dimension.of the howmany_rank-dimensional space mapped by the transforms to be computed.
- int is – the number of memory elements, of type fftw_complex, between the start of transforms in the i th dimension. of the howmany_rank-dimensional space mapped by the transforms to be computed.
- int os – the stride between successive elements in the output vector in the i th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the howmany_dims.n[] entries. For example, if howmany_rank=1, then howmany_dims[0].n is the memory distance between successive transforms.

Constraint: howmany_dims.n[i] >= 1 for i in 0, howmany_rank-1

ri Input parameter

ri is double *

ri is a pointer to a previously allocated array of double data. At fftw_execute() time this will have been filled with the real parts of the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

ii Input parameter

ii is double *

ii is a pointer to a previously allocated array of double data. At fftw_execute() time this will have been filled with the imaginary parts of the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

ro Input parameter

ro is double *

ro is a pointer to a previously allocated array of double data. At fftw_execute() time this will be filled with the real parts of the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

io Input parameter

io is double *

io is a pointer to a previously allocated array of double data. At fftw_execute() time this will be filled with the imaginary parts of the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

sign Input parameter

sign is int

sign is an integer denoting whether a forward or backward FFT transform should be performed

Constraint: sign = -1 (FFTW_FORWARD) for a forward transform,

sign = 1 (FFTW_BACKWARD) for a backwards transform

flags Input parameter

flags is unsigned

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in flags are:

- FFTW_ESTIMATE – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- FFTW_MEASURE – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- FFTW_PATIENT – Produce a plan, testing more options than FFTW_MEASURE. Input and output data might be overwritten.
- FFTW_EXHAUSTIVE – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- FFTW_WISDOM_ONLY – Produce a plan only if the requested case has existing wisdom, otherwise return NULL.
- FFTW_PRESERVE_INPUT – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- FFTW_DESTROY_INPUT – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- FFTW_CONSERVE_MEMORY – Request that extra memory is not created. This might be ignored.
- FFTW_UNALIGNED – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftw_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.3.7 fftw_plan_guru64_dft

`fftw_plan_guru64_dft` is a guru FFTW interface call for planning an *n*-dimensional complex-to-complex FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner.

Unlike *fftw_plan_guru_dft*, `fftw_plan_guru64_dft` allows selected parameters to be specified as 64-bit integers. For Arm Performance Libraries, this parameter size difference is true in both `armpl*_lp64` and `armpl_ilp64` variants. Therefore, the parameter size difference is independent of the integer size used in the rest of the library interface.

Syntax

C specification:

```
#include "fftw3.h"

fftw_plan fftw_plan_guru64_dft(int rank, const fftw_iodim64 *dims,
                               int howmany_rank,
                               const fftw_iodim64 *howmany_dims,
                               fftw_complex *in, fftw_complex *out,
                               int sign, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_plan_guru64_dft(rank,dims,howmany_rank, &
    howmany_dims,in,out,sign,flags) &
    bind(C, name='fftw_plan_guru64_dft')
  integer(C_INT), value :: rank
  type(fftw_iodim64), dimension(*), intent(in) :: dims
  integer(C_INT), value :: howmany_rank
  type(fftw_iodim64), dimension(*), intent(in) :: howmany_dims
  complex(C_DOUBLE_COMPLEX), dimension(*), intent(out) :: in
  complex(C_DOUBLE_COMPLEX), dimension(*), intent(out) :: out
  integer(C_INT), value :: sign
  integer(C_INT), value :: flags
end function fftw_plan_guru64_dft
```

Returns

An `fftw_plan` object is returned that can be used in subsequent *fftw_execute*`()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftw_execute_dft*`()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

`dims` Input parameter

`dims` is `const fftw_iodim64 *`

`dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `ptrdiff_t n` – the length of the `i` th dimension.
- `ptrdiff_t is` – the stride between successive elements in the input vector in the `i` th dimension.
- `ptrdiff_t os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: `dims.n[i]` >= 1 for i in 0, `rank-1`

`howmany_rank` Input parameter

`howmany_rank` is `int`

`howmany_rank` is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then `howmany_rank` needs only be 1.

Constraint: `howmany` >= 0

Note: If `howmany_rank=0`, then a single FFT transform is computed.

`howmany_dims` Input parameter

`howmany_dims` is `const fftw_iodim64 *`

`howmany_dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `ptrdiff_t n` – the number of transforms to be computed of the `i` th dimension.of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `ptrdiff_t is` – the number of memory elements, of type `fftw_complex`, between the start of transforms in the `i` th dimension. of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `ptrdiff_t os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the `howmany_dims.n[]` entries. For example, if `howmany_rank=1`, then `howmany_dims[0].n` is the memory distance between successive transforms.

Constraint: `howmany_dims.n[i]` >= 1 for i in 0, `howmany_rank-1`

`in` Input parameter

`in` is `fftw_complex*`

`in` is a pointer to a previously allocated array of `fftw_complex` data. At `fftw_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `fftw_complex*`

`out` is a pointer to a previously allocated array of `fftw_complex` data. At `fftw_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

sign Input parameter

`sign` is `int`

`sign` is an integer denoting whether a forward or backward FFT transform should be performed

Constraint: `sign` = -1 (`FFTW_FORWARD`) for a forward transform,

`sign` = 1 (`FFTW_BACKWARD`) for a backwards transform

flags Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftw_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.3.8 fftw_plan_guru64_split_dft

`fftw_plan_guru64_split_dft` is a guru FFTW interface call for planning an *n*-dimensional complex-to-complex FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner. Input and output data is stored in separate arrays for real and imaginary parts.

Unlike *fftw_plan_guru_split_dft*, `fftw_plan_guru64_split_dft` allows selected parameters to be specified as 64-bit integers. For Arm Performance Libraries, this parameter size difference is true in both `armpl*_lp64` and `armpl_ilp64` variants. Therefore, the parameter size difference is independent of the integer size used in the rest of the library interface.

Syntax

C specification:

```
#include "fftw3.h"

fftw_plan fftw_plan_guru_split_dft(int rank, const fftw_iodim64 *dims,
                                   int howmany_rank,
                                   const fftw_iodim64 *howmany_dims,
                                   double *ri, double *ii, double *ro,
                                   double *io, int sign, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_plan_guru64_split_dft(rank,dims,howmany_rank, &
    howmany_dims,ri,ii,ro,io,flags) &
    bind(C, name='fftw_plan_guru64_split_dft')
  integer(C_INT), value :: rank
  type(fftw_iodim64), dimension(*), intent(in) :: dims
  integer(C_INT), value :: howmany_rank
  type(fftw_iodim64), dimension(*), intent(in) :: howmany_dims
  real(C_DOUBLE), dimension(*), intent(out) :: ri
  real(C_DOUBLE), dimension(*), intent(out) :: ii
  real(C_DOUBLE), dimension(*), intent(out) :: ro
  real(C_DOUBLE), dimension(*), intent(out) :: io
  integer(C_INT), value :: flags
end function fftw_plan_guru64_split_dft
```

Returns

An `fftw_plan` object is returned that can be used in subsequent *fftw_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftw_execute_dft* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

`dims` Input parameter

`dims` is `const fftw_iodim64 *`

`dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `ptrdiff_t n` – the length of the `i` th dimension.
- `ptrdiff_t is` – the stride between successive elements in the input vector in the `i` th dimension.
- `ptrdiff_t os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: `dims.n[i]` >= 1 for i in 0, `rank-1`

howmany_rank Input parameter

howmany_rank is int

howmany_rank is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then howmany_rank needs only be 1.

Constraint: howmany >= 0

Note: If howmany_rank=0, then a single FFT transform is computed.

howmany_dims Input parameter

howmany_dims is const fftw_iodim64 *

howmany_dims is an array of structures, dimension [rank], where the members of dims[i], for i in 0, rank-1, are defined to be:

- ptrdiff_t n – the number of transforms to be computed of the i th dimension.of the howmany_rank-dimensional space mapped by the transforms to be computed.
- ptrdiff_t is – the number of memory elements, of type fftw_complex, between the start of transforms in the i th dimension. of the howmany_rank-dimensional space mapped by the transforms to be computed.
- ptrdiff_t os – the stride between successive elements in the output vector in the i th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the howmany_dims.n[] entries. For example, if howmany_rank=1, then howmany_dims[0].n is the memory distance between successive transforms.

Constraint: howmany_dims.n[i] >= 1 for i in 0, howmany_rank-1

ri Input parameter

ri is double *

ri is a pointer to a previously allocated array of double data. At fftw_execute() time this will have been filled with the real parts of the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

ii Input parameter

ri is double *

ri is a pointer to a previously allocated array of double data. At fftw_execute() time this will have been filled with the imaginary parts of the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

ro Input parameter

ro is double *

ro is a pointer to a previously allocated array of double data. At fftw_execute() time this will be filled with the real parts of the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

io Input parameter

io is double *

io is a pointer to a previously allocated array of double data. At fftw_execute() time this will be filled with the imaginary parts of the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

sign Input parameter

`sign` is `int`

`sign` is an integer denoting whether a forward or backward FFT transform should be performed

Constraint: `sign` = -1 (`FFTW_FORWARD`) for a forward transform,

`sign` = 1 (`FFTW_BACKWARD`) for a backwards transform

flags Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftw_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.3.9 fftw_plan_many_dft

`fftw_plan_many_dft` is an advanced FFTW interface call for planning an *n*-dimensional complex-to-complex FFT operation on `howmany` sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner.

For more information on the use of advanced planning calls see *How to choose which FFT routine you need*.

Syntax

C specification:

```
#include "fftw3.h"

fftw_plan fftw_plan_many_dft(int rank, const int *n, int howmany,
                             fftw_complex *in, const int *inembed,
                             int istride, int idist, fftw_complex *out,
                             const int *onembed, int ostride,
                             int odist, int sign, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_plan_many_dft(rank,n,howmany,in,inembed, &
    istride,idist,out,onembed,ostride,odist,sign,flags) &
    bind(C, name='fftw_plan_many_dft')
  integer(C_INT), value :: rank
  integer(C_INT), dimension(*), intent(in) :: n
  integer(C_INT), value :: howmany
  complex(C_DOUBLE_COMPLEX), dimension(*), intent(out) :: in
  integer(C_INT), dimension(*), intent(in) :: inembed
  integer(C_INT), value :: istride
  integer(C_INT), value :: idist
  complex(C_DOUBLE_COMPLEX), dimension(*), intent(out) :: out
  integer(C_INT), dimension(*), intent(in) :: onembed
  integer(C_INT), value :: ostride
  integer(C_INT), value :: odist
  integer(C_INT), value :: sign
  integer(C_INT), value :: flags
end function fftw_plan_many_dft
```

Returns

An `fftw_plan` object is returned that can be used in subsequent *fftw_execute*`()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftw_execute_dft*`()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of the FFT problem to be solved

Constraint: `rank` >= 1

`n` Input parameter

`n` is `const int*`

`n` is an array of dimension `[rank]`.

`n` contains the size of each dimension of the FFT to be solved

Constraint: `n[i]` >= 1 for i in 0, `rank-1`

`howmany` Input parameter

`howmany` is `int`

`howmany` is the number of multi-dimensional FFT transforms to be computed

Constraint: `howmany` >= 1

in Input parameter

in is fftw_complex*

in is a pointer to a previously allocated array of fftw_complex data. At fftw_execute() time this will have been filled with the input sequence on which to perform the FFT.. The input data for set k starts at address in+k*idist.

Storage: Data in arrays will be stored in row-major order.

inembed Input parameter

inembed is const int

inembed is an array of dimension [rank] or NULL This array denotes the dimensionality of a potentially larger space of which the input array, in, dimensions given by n, is a sub array.

Constraint: Either inembed[i] >= n[i] for i in 0, rank-1 OR if inembed == NULL, then inembed is assumed to have the same dimensions as n

istride Input parameter

istride is int

istride is the distance between successive elements of the input data

idist Input parameter

idist is int

idist is the distance between the start of each of the howmany transforms in the input data

out Input parameter

out is fftw_complex*

out is a pointer to a previously allocated array of fftw_complex data. At fftw_execute() time this will be filled with the resulting transform coefficients. The output data for set k starts at address out+k*odist.

Storage: Data in arrays will be stored in row-major order.

onembed Input parameter

onembed is const int

onembed is an array of dimension [rank] or NULL. This array denotes the dimensionality of a potentially larger space of which the input array, out, dimensions given by n, is a sub array.

Constraint: Either onembed[i] >= n[i] for i in 0, rank-1 OR if onembed == NULL, then onembed is assumed to have the same dimensions as n

ostride Input parameter

ostride is int

ostride is the distance between successive elements of the output data

odist Input parameter

odist is int

odist is the distance between the start of each of the howmany transforms in the output data

sign Input parameter

sign is int

sign is an integer denoting whether a forward or backward FFT transform should be performed

Constraint: sign = -1 (FFTW_FORWARD) for forward transform,

sign = 1 (FFTW_BACKWARD) for backwards transform

flags Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftw_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.3.10 fftwf_plan_dft

`fftwf_plan_dft` is a basic FFTW interface call for planning an *n*-dimensional complex-to-complex FFT operation on a single, contiguous data sequence.

Note: `in` and `out` can be the same pointer for an "in-place" transform.

Syntax

C specification:

```
#include "fftw3.h"

fftwf_plan fftwf_plan_dft(int rank, const int *n, fftwf_complex *in,
                          fftwf_complex *out, int sign, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_plan_dft(rank,n,in,out,sign,flags) &
    bind(C, name='fftwf_plan_dft')
  integer(C_INT), value :: rank
  integer(C_INT), dimension(*), intent(in) :: n
  complex(C_FLOAT_COMPLEX), dimension(*), intent(out) :: in
  complex(C_FLOAT_COMPLEX), dimension(*), intent(out) :: out
  integer(C_INT), value :: sign
  integer(C_INT), value :: flags
end function fftwf_plan_dft
```

Returns

This function returns an `fftwf_plan` object that can be used in subsequent *fftwf_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwf_execute_dft* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of the FFT problem to be solved

Constraint: `rank` >= 1

`n` Input parameter

`n` is `const int*`.

`n` is an array of dimension `[rank]`.

`n` contains the size of each dimension of the FFT to be solved

Constraint: `n[i]` >= 1 for i in 0, `rank-1`

`in` Input parameter

`in` is `fftwf_complex*`

`in` is a pointer to a previously allocated array of `fftwf_complex` data. At `fftwf_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `fftwf_complex*`

`out` is a pointer to a previously allocated array of `fftwf_complex` data. At `fftwf_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`sign` Input parameter

`sign` is `int`

`sign` is an integer denoting whether a forward or backward FFT transform should be performed

Constraint: `sign` = -1 (`FFTW_FORWARD`) for a forward transform,

`sign` = 1 (`FFTW_BACKWARD`) for a backwards transform

flags Input parameter

flags is unsigned

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in flags are:

- FFTW_ESTIMATE – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- FFTW_MEASURE – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- FFTW_PATIENT – Produce a plan, testing more options than FFTW_MEASURE. Input and output data might be overwritten.
- FFTW_EXHAUSTIVE – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- FFTW_WISDOM_ONLY – Produce a plan only if the requested case has existing wisdom, otherwise return NULL.
- FFTW_PRESERVE_INPUT – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- FFTW_DESTROY_INPUT – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- FFTW_CONSERVE_MEMORY – Request that extra memory is not created. This might be ignored.
- FFTW_UNALIGNED – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by FFTW_MEASURE, FFTW_PATIENT and FFTW_EXHAUSTIVE will be limited when using *fftwf_set_timelimit*.
- If multiple options for planning rigor are included in flags, then the highest level of planning requested will be used.

5.3.11 fftwf_plan_dft_1d

fftwf_plan_dft_1d is a basic FFTW interface call for planning a 1-d complex-to-complex FFT operation on a single, contiguous data sequence.

Note: in and out can be the same pointer for an "in-place" transform.

Syntax

C specification:

```
#include "fftw3.h"

fftwf_plan fftwf_plan_dft_1d(int n0, fftwf_complex *in, fftwf_complex *out,
                             int sign, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_plan_dft_1d(n,in,out,sign,flags) &
    bind(C, name='fftwf_plan_dft_1d')
  integer(C_INT), value :: n
  complex(C_FLOAT_COMPLEX), dimension(*), intent(out) :: in
  complex(C_FLOAT_COMPLEX), dimension(*), intent(out) :: out
  integer(C_INT), value :: sign
  integer(C_INT), value :: flags
end function fftwf_plan_dft_1d
```

Returns

This function returns an `fftwf_plan` object that can be used in subsequent *fftwf_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwf_execute_dft* `()`.

Parameters

`n0` Input parameter

`n0` is `int`

This is the length of the FFT sequence

Constraint: `n0` >= 1

`in` Input parameter

`in` is `fftwf_complex*`

`in` is a pointer to a previously allocated array of `fftwf_complex` data. At `fftwf_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `fftwf_complex*`

`out` is a pointer to a previously allocated array of `fftwf_complex` data. At `fftwf_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`sign` Input parameter

`sign` is `int`

`sign` is an integer denoting whether a forward or backward FFT transform should be performed

Constraint: `sign` = -1 (`FFTW_FORWARD`) for a forward transform,

`sign` = 1 (`FFTW_BACKWARD`) for a backwards transform

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.

- FFTW_PATIENT – Produce a plan, testing more options than FFTW_MEASURE. Input and output data might be overwritten.
- FFTW_EXHAUSTIVE – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- FFTW_WISDOM_ONLY – Produce a plan only if the requested case has existing wisdom, otherwise return NULL.
- FFTW_PRESERVE_INPUT – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- FFTW_DESTROY_INPUT – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- FFTW_CONSERVE_MEMORY – Request that extra memory is not created. This might be ignored.
- FFTW_UNALIGNED – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by FFTW_MEASURE, FFTW_PATIENT and FFTW_EXHAUSTIVE will be limited when using *fftwf_set_timelimit*.
- If multiple options for planning rigor are included in flags, then the highest level of planning requested will be used.

5.3.12 fftwf_plan_dft_2d

fftwf_plan_dft_2d is a basic FFTW interface call for planning a 2-d complex-to-complex FFT operation on a single, contiguous data sequence.

Note: in and out can be the same pointer for an "in-place" transform.

Syntax

C specification:

```
#include "fftw3.h"

fftwf_plan fftwf_plan_dft_2d(int n0, int n1, fftwf_complex *in,
                             fftwf_complex *out, int sign, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_plan_dft_2d(n0,n1,in,out,sign,flags) &
    bind(C, name='fftwf_plan_dft_2d')
  integer(C_INT), value :: n0
  integer(C_INT), value :: n1
  complex(C_FLOAT_COMPLEX), dimension(*), intent(out) :: in
  complex(C_FLOAT_COMPLEX), dimension(*), intent(out) :: out
  integer(C_INT), value :: sign
  integer(C_INT), value :: flags
end function fftwf_plan_dft_2d
```

Returns

This function returns an `fftwf_plan` object that can be used in subsequent *fftwf_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwf_execute_dft* `()`.

Parameters

`n0` Input parameter

`n0` is `int`

This is the length of the first dimension of the FFT sequence

Constraint: `n0` >= 1

`n1` Input parameter

`n1` is `int`

This is the length of the second dimension of the FFT sequence

Constraint: `n1` >= 1

`in` Input parameter

`in` is `fftwf_complex*`

`in` is a pointer to a previously allocated array of `fftwf_complex` data. At `fftwf_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

out is fftwf_complex

out is a pointer to a previously allocated array of fftwf_complex data. At fftwf_execute() time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`sign` Input parameter

`sign` is `int`

`sign` is an integer denoting whether a forward or backward FFT transform should be performed

Constraint: `sign` = -1 (`FFTW_FORWARD`) for a forward transform,

`sign` = 1 (`FFTW_BACKWARD`) for a backwards transform

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.

- FFTW_WISDOM_ONLY – Produce a plan only if the requested case has existing wisdom, otherwise return NULL.
- FFTW_PRESERVE_INPUT – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- FFTW_DESTROY_INPUT – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- FFTW_CONSERVE_MEMORY – Request that extra memory is not created. This might be ignored.
- FFTW_UNALIGNED – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by FFTW_MEASURE, FFTW_PATIENT and FFTW_EXHAUSTIVE will be limited when using *fftwf_set_timelimit*.
- If multiple options for planning rigor are included in flags, then the highest level of planning requested will be used.

5.3.13 fftwf_plan_dft_3d

fftwf_plan_dft_3d is a basic FFTW interface call for planning a 3-d complex-to-complex FFT operation on a single, contiguous data sequence.

Note: in and out can be the same pointer for an "in-place" transform.

Syntax

C specification:

```
#include "fftw3.h"

fftwf_plan fftwf_plan_dft_3d(int n0, int n1, int n2, fftwf_complex *in,
                             fftwf_complex *out, int sign, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_plan_dft_3d(n0,n1,n2,in,out,sign,flags) &
    bind(C, name='fftwf_plan_dft_3d')
  integer(C_INT), value :: n0
  integer(C_INT), value :: n1
  integer(C_INT), value :: n2
  complex(C_FLOAT_COMPLEX), dimension(*), intent(out) :: in
  complex(C_FLOAT_COMPLEX), dimension(*), intent(out) :: out
  integer(C_INT), value :: sign
  integer(C_INT), value :: flags
end function fftwf_plan_dft_3d
```

Returns

This function returns an fftwf_plan object that can be used in subsequent *fftwf_execute*() calls on the previously supplied in and out arrays, or using different arrays via *fftwf_execute_dft*().

Parameters

n0 Input parameter

`n0` is `int`

This is the length of the first dimension of the FFT sequence

Constraint: `n0` >= 1

n1 Input parameter

`n1` is `int`

This is the length of the second dimension of the FFT sequence

Constraint: `n1` >= 1

n2 Input parameter

`n2` is `int`

This is the length of the third dimension of the FFT sequence

Constraint: `n2` >= 1

in Input parameter

`in` is `fftwf_complex*`

`in` is a pointer to a previously allocated array of `fftwf_complex` data. At `fftwf_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

out Input parameter

`out` is `fftwf_complex*`

`out` is a pointer to a previously allocated array of `fftwf_complex` data. At `fftwf_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

sign Input parameter

`sign` is `int`

`sign` is an integer denoting whether a forward or backward FFT transform should be performed

Constraint: `sign` = -1 (`FFTW_FORWARD`) for a forward transform,

`sign` = 1 (`FFTW_BACKWARD`) for a backwards transform

flags Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.

- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwf_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.3.14 fftwf_plan_guru_dft

`fftwf_plan_guru_dft` is a guru FFTW interface call for planning an *n*-dimensional complex-to-complex FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner.

Syntax

C specification:

```
#include "fftw3.h"

fftwf_plan fftwf_plan_guru_dft(int rank, const fftwf_iodim *dims,
                               int howmany_rank,
                               const fftwf_iodim *howmany_dims,
                               fftwf_complex *in, fftwf_complex *out,
                               int sign, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_plan_guru_dft(rank,dims,howmany_rank, &
    howmany_dims,in,out,sign,flags) &
    bind(C, name='fftwf_plan_guru_dft')
  integer(C_INT), value :: rank
  type(fftwf_iodim), dimension(*), intent(in) :: dims
  integer(C_INT), value :: howmany_rank
  type(fftwf_iodim), dimension(*), intent(in) :: howmany_dims
  complex(C_FLOAT_COMPLEX), dimension(*), intent(out) :: in
  complex(C_FLOAT_COMPLEX), dimension(*), intent(out) :: out
  integer(C_INT), value :: sign
  integer(C_INT), value :: flags
end function fftwf_plan_guru_dft
```

Returns

This function returns an `fftwf_plan` object that can be used in subsequent *fftwf_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwf_execute_dft* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

`dims` Input parameter

`dims` is `const fftwf_iodim *`

`dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the length of the `i` th dimension.
- `int is` – the stride between successive elements in the input vector in the `i` th dimension.
- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: `dims.n[i]` >= 1 for i in 0, `rank-1`

`howmany_rank` Input parameter

`howmany_rank` is `int`

`howmany_rank` is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then `howmany_rank` needs only be 1.

Constraint: `howmany` >= 0

Note: If `howmany_rank=0`, then a single FFT transform is computed.

`howmany_dims` Input parameter

`howmany_dims` is `const fftwf_iodim *`

`howmany_dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the number of transforms to be computed of the `i` th dimension.of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `int is` – the number of memory elements, of type `fftwf_complex`, between the start of transforms in the `i` th dimension. of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the `howmany_dims.n[]` entries. For example, if `howmany_rank=1`, then `howmany_dims[0].n` is the memory distance between successive transforms.

Constraint: `howmany_dims.n[i]` >= 1 for i in 0, `howmany_rank-1`

in Input parameter

in is fftwf_complex*

in is a pointer to a previously allocated array of fftwf_complex data. At fftwf_execute() time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

out Input parameter

out is fftwf_complex*

out is a pointer to a previously allocated array of fftwf_complex data. At fftwf_execute() time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

sign Input parameter

sign is int

sign is an integer denoting whether a forward or backward FFT transform should be performed

Constraint: sign = -1 (FFTW_FORWARD) for a forward transform,

sign = 1 (FFTW_BACKWARD) for a backwards transform

flags Input parameter

flags is unsigned

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in flags are:

- FFTW_ESTIMATE – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- FFTW_MEASURE – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- FFTW_PATIENT – Produce a plan, testing more options than FFTW_MEASURE. Input and output data might be overwritten.
- FFTW_EXHAUSTIVE – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- FFTW_WISDOM_ONLY – Produce a plan only if the requested case has existing wisdom, otherwise return NULL.
- FFTW_PRESERVE_INPUT – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- FFTW_DESTROY_INPUT – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- FFTW_CONSERVE_MEMORY – Request that extra memory is not created. This might be ignored.
- FFTW_UNALIGNED – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by FFTW_MEASURE, FFTW_PATIENT and FFTW_EXHAUSTIVE will be limited when using *fftwf_set_timelimit*.
- If multiple options for planning rigor are included in flags, then the highest level of planning requested will be used.

5.3.15 fftwf_plan_guru_split_dft

`fftwf_plan_guru_split_dft` is a guru FFTW interface call for planning an *n*-dimensional complex-to-complex FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner. Input and output data is stored in separate arrays for real and imaginary parts.

Syntax

C specification:

```
#include "fftw3.h"

fftwf_plan fftwf_plan_guru_split_dft(int rank, const fftwf_iodim *dims,
                                     int howmany_rank,
                                     const fftwf_iodim *howmany_dims,
                                     float *ri, float *ii, float *ro,
                                     float *io, int sign, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_plan_guru_split_dft(rank,dims,howmany_rank, &
    howmany_dims,ri,ii,ro,io,flags) &
    bind(C, name='fftwf_plan_guru_split_dft')
  integer(C_INT), value :: rank
  type(fftwf_iodim), dimension(*), intent(in) :: dims
  integer(C_INT), value :: howmany_rank
  type(fftwf_iodim), dimension(*), intent(in) :: howmany_dims
  real(C_FLOAT), dimension(*), intent(out) :: ri
  real(C_FLOAT), dimension(*), intent(out) :: ii
  real(C_FLOAT), dimension(*), intent(out) :: ro
  real(C_FLOAT), dimension(*), intent(out) :: io
  integer(C_INT), value :: flags
end function fftwf_plan_guru_split_dft
```

Returns

This function returns an `fftwf_plan` object that can be used in subsequent *fftwf_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwf_execute_dft* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

`dims` Input parameter

`dims` is `const fftwf_iodim *`

`dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the length of the `i` th dimension.
- `int is` – the stride between successive elements in the input vector in the `i` th dimension.

- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: `dims.n[i]` >= 1 for i in 0, `rank-1`

howmany_rank Input parameter

`howmany_rank` is `int`

`howmany_rank` is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then `howmany_rank` needs only be 1.

Constraint: `howmany` >= 0

Note: If `howmany_rank=0`, then a single FFT transform is computed.

howmany_dims Input parameter

`howmany_dims` is `const fftwf_iodim *`

`howmany_dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the number of transforms to be computed of the `i` th dimension.of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `int is` – the number of memory elements, of type `fftwf_complex`, between the start of transforms in the `i` th dimension. of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the `howmany_dims.n[]` entries. For example, if `howmany_rank=1`, then `howmany_dims[0].n` is the memory distance between successive transforms.

Constraint: `howmany_dims.n[i]` >= 1 for i in 0, `howmany_rank-1`

ri Input parameter

`ri` is `float *`

`ri` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will have been filled with the real parts of the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

ii Input parameter

`ii` is `float *`

`ii` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will have been filled with the imaginary parts of the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

ro Input parameter

`ro` is `float *`

`ro` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will be filled with the real parts of the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

io Input parameter

`io` is `float *`

`io` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will be filled with the imaginary parts of the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`sign` Input parameter

`sign` is `int`

`sign` is an integer denoting whether a forward or backward FFT transform should be performed

Constraint: `sign` = -1 (`FFTW_FORWARD`) for a forward transform,

`sign` = 1 (`FFTW_BACKWARD`) for a backwards transform

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwf_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.3.16 fftwf_plan_guru64_dft

`fftwf_plan_guru64_dft` is a guru FFTW interface call for planning an *n*-dimensional complex-to-complex FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner.

Unlike *fftwf_plan_guru_dft*, `fftwf_plan_guru64_dft` allows selected parameters to be specified as 64-bit integers. For Arm Performance Libraries, this parameter size difference is true in both `armpl*_lp64` and `armpl_ilp64` variants. Therefore, the parameter size difference is independent of the integer size used in the rest of the library interface.

Syntax

C specification:

```
#include "fftw3.h"

fftwf_plan fftwf_plan_guru64_dft(int rank, const fftwf_iodim64 *dims,
                                 int howmany_rank,
                                 const fftwf_iodim64 *howmany_dims,
                                 fftwf_complex *in, fftwf_complex *out,
                                 int sign, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_plan_guru64_dft(rank,dims,howmany_rank, &
    howmany_dims,in,out,sign,flags) &
    bind(C, name='fftwf_plan_guru64_dft')
  integer(C_INT), value :: rank
  type(fftwf_iodim64), dimension(*), intent(in) :: dims
  integer(C_INT), value :: howmany_rank
  type(fftwf_iodim64), dimension(*), intent(in) :: howmany_dims
  complex(C_FLOAT_COMPLEX), dimension(*), intent(out) :: in
  complex(C_FLOAT_COMPLEX), dimension(*), intent(out) :: out
  integer(C_INT), value :: sign
  integer(C_INT), value :: flags
end function fftwf_plan_guru64_dft
```

Returns

This function returns an `fftwf_plan` object that can be used in subsequent *fftwf_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwf_execute_dft* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

`dims` Input parameter

`dims` is `const fftwf_iodim64 *`

`dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `ptrdiff_t n` – the length of the `i` th dimension.
- `ptrdiff_t is` – the stride between successive elements in the input vector in the `i` th dimension.
- `ptrdiff_t os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: `dims.n[i]` >= 1 for i in 0, `rank-1`

howmany_rank Input parameter

howmany_rank is int

howmany_rank is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then howmany_rank needs only be 1.

Constraint: howmany >= 0

Note: If howmany_rank=0, then a single FFT transform is computed.

howmany_dims Input parameter

howmany_dims is const fftwf_iodim64 *

howmany_dims is an array of structures, dimension [rank], where the members of dims[i], for i in 0, rank-1, are defined to be:

- ptrdiff_t n – the number of transforms to be computed of the i th dimension.of the howmany_rank-dimensional space mapped by the transforms to be computed.
- ptrdiff_t is – the number of memory elements, of type fftwf_complex, between the start of transforms in the i th dimension. of the howmany_rank-dimensional space mapped by the transforms to be computed.
- ptrdiff_t os – the stride between successive elements in the output vector in the i th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the howmany_dims.n[] entries. For example, if howmany_rank=1, then howmany_dims[0].n is the memory distance between successive transforms.

Constraint: howmany_dims.n[i] >= 1 for i in 0, howmany_rank-1

in Input parameter

in is fftwf_complex*

in is a pointer to a previously allocated array of fftwf_complex data. At fftwf_execute() time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

out Input parameter

out is fftwf_complex*

out is a pointer to a previously allocated array of fftwf_complex data. At fftwf_execute() time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

sign Input parameter

sign is int

sign is an integer denoting whether a forward or backward FFT transform should be performed

Constraint: sign = -1 (FFTW_FORWARD) for a forward transform,

sign = 1 (FFTW_BACKWARD) for a backwards transform

flags Input parameter

flags is unsigned

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in flags are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwf_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.3.17 fftwf_plan_guru64_split_dft

`fftwf_plan_guru64_split_dft` is a guru FFTW interface call for planning an *n*-dimensional complex-to-complex FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner. Input and output data is stored in separate arrays for real and imaginary parts.

Unlike *fftwf_plan_guru_split_dft*, `fftwf_plan_guru64_split_dft` allows selected parameters to be specified as 64-bit integers. For Arm Performance Libraries, this parameter size difference is true in both `armpl*_lp64` and `armpl_ilp64` variants. Therefore, the parameter size difference is independent of the integer size used in the rest of the library interface.

Syntax

C specification:

```
#include "fftw3.h"

fftwf_plan fftwf_plan_guru_split_dft(int rank, const fftwf_iodim64 *dims,
                                     int howmany_rank,
                                     const fftwf_iodim64 *howmany_dims,
                                     float *ri, float *ii, float *ro,
                                     float *io, int sign, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'
```

(continues on next page)

(continued from previous page)

```
type(C_PTR) function fftwf_plan_guru64_split_dft(rank,dims, &
    howmany_rank,howmany_dims,ri,ii,ro,io,flags) &
    bind(C, name='fftwf_plan_guru64_split_dft')
  integer(C_INT), value :: rank
  type(fftwf_iodim64), dimension(*), intent(in) :: dims
  integer(C_INT), value :: howmany_rank
  type(fftwf_iodim64), dimension(*), intent(in) :: howmany_dims
  real(C_FLOAT), dimension(*), intent(out) :: ri
  real(C_FLOAT), dimension(*), intent(out) :: ii
  real(C_FLOAT), dimension(*), intent(out) :: ro
  real(C_FLOAT), dimension(*), intent(out) :: io
  integer(C_INT), value :: flags
end function fftwf_plan_guru64_split_dft
```

Returns

This function returns an `fftwf_plan` object that can be used in subsequent *fftwf_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwf_execute_dft* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

`dims` Input parameter

`dims` is `const fftwf_iodim64 *`

`dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `ptrdiff_t n` – the length of the `i` th dimension.
- `ptrdiff_t is` – the stride between successive elements in the input vector in the `i` th dimension.
- `ptrdiff_t os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: `dims.n[i]` >= 1 for i in 0, `rank-1`

`howmany_rank` Input parameter

`howmany_rank` is `int`

`howmany_rank` is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then `howmany_rank` needs only be 1.

Constraint: `howmany` >= 0

Note: If `howmany_rank=0`, then a single FFT transform is computed.

`howmany_dims` Input parameter

`howmany_dims` is `const fftwf_iodim64 *`

`howmany_dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `ptrdiff_t n` – the number of transforms to be computed of the `i` th dimension.of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `ptrdiff_t is` – the number of memory elements, of type `fftwf_complex`, between the start of transforms in the `i` th dimension. of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `ptrdiff_t os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the `howmany_dims.n[]` entries. For example, if `howmany_rank=1`, then `howmany_dims[0].n` is the memory distance between successive transforms.

Constraint: `howmany_dims.n[i]` >= 1 for i in 0, `howmany_rank-1`

ri Input parameter

`ri` is `float *`

`ri` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will have been filled with the real parts of the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

ii Input parameter

`ri` is `float *`

`ri` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will have been filled with the imaginary parts of the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

ro Input parameter

`ro` is `float *`

`ro` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will be filled with the real parts of the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

io Input parameter

`io` is `float *`

`io` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will be filled with the imaginary parts of the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

sign Input parameter

`sign` is `int`

`sign` is an integer denoting whether a forward or backward FFT transform should be performed

Constraint: `sign` = -1 (`FFTW_FORWARD`) for a forward transform,

`sign` = 1 (`FFTW_BACKWARD`) for a backwards transform

flags Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.

- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwf_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.3.18 fftwf_plan_many_dft

`fftwf_plan_many_dft` is an advanced FFTW interface call for planning an *n*-dimensional complex-to-complex FFT operation on `howmany` sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner.

For more information on the use of advanced planning calls see *How to choose which FFT routine you need*.

Syntax

C specification:

```
#include "fftw3.h"

fftwf_plan fftwf_plan_many_dft(int rank, const int *n, int howmany,
                               fftwf_complex *in, const int *inembed,
                               int istride, int idist, fftwf_complex *out,
                               const int *onembed, int ostride,
                               int odist, int sign, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_plan_many_dft(rank,n,howmany,in,inembed, &
    istride,idist,out,onembed,ostride,odist,sign,flags) &
    bind(C, name='fftwf_plan_many_dft')
  integer(C_INT), value :: rank
  integer(C_INT), dimension(*), intent(in) :: n
  integer(C_INT), value :: howmany
```

(continues on next page)

(continued from previous page)

```
  complex(C_FLOAT_COMPLEX), dimension(*), intent(out) :: in
  integer(C_INT), dimension(*), intent(in) :: inembed
  integer(C_INT), value :: istride
  integer(C_INT), value :: idist
  complex(C_FLOAT_COMPLEX), dimension(*), intent(out) :: out
  integer(C_INT), dimension(*), intent(in) :: onembed
  integer(C_INT), value :: ostride
  integer(C_INT), value :: odist
  integer(C_INT), value :: sign
  integer(C_INT), value :: flags
end function fftwf_plan_many_dft
```

Returns

This function returns an `fftwf_plan` object that can be used in subsequent *fftwf_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwf_execute_dft* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of the FFT problem to be solved

Constraint: `rank` >= 1

`n` Input parameter

`n` is `const int*`

`n` is an array of dimension `[rank]`.

`n` contains the size of each dimension of the FFT to be solved

Constraint: `n[i]` >= 1 for i in 0, `rank-1`

`howmany` Input parameter

`howmany` is `int`

`howmany` is the number of multi-dimensional FFT transforms to be computed

Constraint: `howmany` >= 1

`in` Input parameter

`in` is `fftwf_complex*`

`in` is a pointer to a previously allocated array of `fftwf_complex` data. At `fftwf_execute()` time this will have been filled with the input sequence on which to perform the FFT.. The input data for set `k` starts at address `in+k*idist`.

Storage: Data in arrays will be stored in row-major order.

`inembed` Input parameter

`inembed` is `const int`

`inembed` is an array of dimension `[rank]` or `NULL` This array denotes the dimensionality of a potentially larger space of which the input array, `in`, dimensions given by `n`, is a sub array.

Constraint: Either `inembed[i]` >= `n[i]` for `i` in 0, `rank-1` OR if `inembed` == `NULL`, then `inembed` is assumed to have the same dimensions as `n`

istride Input parameter

istride is int

istride is the distance between successive elements of the input data

idist Input parameter

idist is int

idist is the distance between the start of each of the howmany transforms in the input data

out Input parameter

out is fftwf_complex*

out is a pointer to a previously allocated array of fftwf_complex data. At fftwf_execute() time this will be filled with the resulting transform coefficients. The output data for set k starts at address out+k*odist.

Storage: Data in arrays will be stored in row-major order.

onembed Input parameter

onembed is const int

onembed is an array of dimension [rank] or NULL. This array denotes the dimensionality of a potentially larger space of which the input array, out, dimensions given by n, is a sub array.

Constraint: Either onembed[i] >= n[i] for i in 0, rank-1 OR if onembed == NULL, then onembed is assumed to have the same dimensions as n

ostride Input parameter

ostride is int

ostride is the distance between successive elements of the output data

odist Input parameter

odist is int

odist is the distance between the start of each of the howmany transforms in the output data

sign Input parameter

sign is int

sign is an integer denoting whether a forward or backward FFT transform should be performed

Constraint: sign = -1 (FFTW_FORWARD) for forward transform,

sign = 1 (FFTW_BACKWARD) for backwards transform

flags Input parameter

flags is unsigned

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in flags are:

- FFTW_ESTIMATE – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- FFTW_MEASURE – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- FFTW_PATIENT – Produce a plan, testing more options than FFTW_MEASURE. Input and output data might be overwritten.
- FFTW_EXHAUSTIVE – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.

- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwf_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.3.19 fftwh_plan_dft

`fftwh_plan_dft` is a basic FFTW interface call for planning an *n*-dimensional complex-to-complex FFT operation on a single, contiguous data sequence.

Note: `in` and `out` can be the same pointer for an "in-place" transform.

Syntax

C specification:

```
#include "fftw3.h"

fftwh_plan fftwh_plan_dft(int rank, const int *n, fftwh_complex *in,
                          fftwh_complex *out, int sign, unsigned flags);
```

Returns

This function returns an `fftwh_plan` object that can be used in subsequent *fftwh_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwh_execute_dft* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of the FFT problem to be solved

Constraint: `rank` >= 1

`n` Input parameter

`n` is `const int*`.

`n` is an array of dimension `[rank]`.

`n` contains the size of each dimension of the FFT to be solved

Constraint: `n[i]` >= 1 for i in 0, `rank-1`

`in` Input parameter

`in` is `fftwh_complex*`

`in` is a pointer to a previously allocated array of `fftwh_complex` data. At `fftwh_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `fftwh_complex*`

`out` is a pointer to a previously allocated array of `fftwh_complex` data. At `fftwh_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`sign` Input parameter

`sign` is `int`

`sign` is an integer denoting whether a forward or backward FFT transform should be performed

Constraint: `sign` = -1 (`FFTW_FORWARD`) for a forward transform,

`sign` = 1 (`FFTW_BACKWARD`) for a backwards transform

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwh_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.3.20 fftwh_plan_dft_1d

`fftwh_plan_dft_1d` is a basic FFTW interface call for planning a 1-d complex-to-complex FFT operation on a single, contiguous data sequence.

Note: `in` and `out` can be the same pointer for an "in-place" transform.

Syntax

C specification:

```
#include "fftw3.h"

fftwh_plan fftwh_plan_dft_1d(int n0, fftwh_complex *in, fftwh_complex *out,
                             int sign, unsigned flags);
```

Returns

This function returns an `fftwh_plan` object that can be used in subsequent *fftwh_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwh_execute_dft* `()`.

Parameters

`n0` Input parameter

`n0` is `int`

This is the length of the FFT sequence

Constraint: `n0` >= 1

`in` Input parameter

`in` is `fftwh_complex*`

`in` is a pointer to a previously allocated array of `fftwh_complex` data. At `fftwh_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `fftwh_complex*`

`out` is a pointer to a previously allocated array of `fftwh_complex` data. At `fftwh_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`sign` Input parameter

`sign` is `int`

`sign` is an integer denoting whether a forward or backward FFT transform should be performed

Constraint: `sign` = -1 (`FFTW_FORWARD`) for a forward transform,

`sign` = 1 (`FFTW_BACKWARD`) for a backwards transform

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwh_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.3.21 fftwh_plan_dft_2d

`fftwh_plan_dft_2d` is a basic FFTW interface call for planning a 2-d complex-to-complex FFT operation on a single, contiguous data sequence.

Note: `in` and `out` can be the same pointer for an "in-place" transform.

Syntax

C specification:

```
#include "fftw3.h"

fftwh_plan fftwh_plan_dft_2d(int n0, int n1, fftwh_complex *in,
                             fftwh_complex *out, int sign, unsigned flags);
```

Returns

This function returns an `fftwh_plan` object that can be used in subsequent *fftwh_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwh_execute_dft* `()`.

Parameters

n0 Input parameter

n0 is int

This is the length of the first dimension of the FFT sequence

Constraint: n0 >= 1

n1 Input parameter

n1 is int

This is the length of the second dimension of the FFT sequence

Constraint: n1 >= 1

in Input parameter

in is fftwh_complex*

in is a pointer to a previously allocated array of fftwh_complex data. At fftwh_execute() time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

out Input parameter

out is fftwh_complex

out is a pointer to a previously allocated array of fftwh_complex data. At fftwh_execute() time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

sign Input parameter

sign is int

sign is an integer denoting whether a forward or backward FFT transform should be performed

Constraint: sign = -1 (FFTW_FORWARD) for a forward transform,

sign = 1 (FFTW_BACKWARD) for a backwards transform

flags Input parameter

flags is unsigned

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in flags are:

- FFTW_ESTIMATE – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- FFTW_MEASURE – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- FFTW_PATIENT – Produce a plan, testing more options than FFTW_MEASURE. Input and output data might be overwritten.
- FFTW_EXHAUSTIVE – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- FFTW_WISDOM_ONLY – Produce a plan only if the requested case has existing wisdom, otherwise return NULL.
- FFTW_PRESERVE_INPUT – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.

- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwh_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.3.22 fftwh_plan_dft_3d

`fftwh_plan_dft_3d` is a basic FFTW interface call for planning a 3-d complex-to-complex FFT operation on a single, contiguous data sequence.

Note: `in` and `out` can be the same pointer for an "in-place" transform.

Syntax

C specification:

```
#include "fftw3.h"

fftwh_plan fftwh_plan_dft_3d(int n0, int n1, int n2, fftwh_complex *in,
                             fftwh_complex *out, int sign, unsigned flags);
```

Returns

This function returns an `fftwh_plan` object that can be used in subsequent *fftwh_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwh_execute_dft* `()`.

Parameters

`n0` Input parameter

`n0` is `int`

This is the length of the first dimension of the FFT sequence

Constraint: `n0` >= 1

`n1` Input parameter

`n1` is `int`

This is the length of the second dimension of the FFT sequence

Constraint: `n1` >= 1

`n2` Input parameter

`n2` is `int`

This is the length of the third dimension of the FFT sequence

Constraint: `n2` >= 1

`in` Input parameter

`in` is `fftwh_complex*`

`in` is a pointer to a previously allocated array of `fftwh_complex` data. At `fftwh_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `fftwh_complex*`

`out` is a pointer to a previously allocated array of `fftwh_complex` data. At `fftwh_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`sign` Input parameter

`sign` is `int`

`sign` is an integer denoting whether a forward or backward FFT transform should be performed

Constraint: `sign` = -1 (`FFTW_FORWARD`) for a forward transform,

`sign` = 1 (`FFTW_BACKWARD`) for a backwards transform

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwh_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.3.23 fftwh_plan_guru_dft

`fftwh_plan_guru_dft` is a guru FFTW interface call for planning an *n*-dimensional complex-to-complex FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner.

Syntax

C specification:

```
#include "fftw3.h"

fftwh_plan fftwh_plan_guru_dft(int rank, const fftwh_iodim *dims,
                               int howmany_rank,
                               const fftwh_iodim *howmany_dims,
                               fftwh_complex *in, fftwh_complex *out,
                               int sign, unsigned flags);
```

Returns

This function returns an `fftwh_plan` object that can be used in subsequent *fftwh_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwh_execute_dft* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

`dims` Input parameter

`dims` is `const fftwh_iodim *`

`dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the length of the `i` th dimension.
- `int is` – the stride between successive elements in the input vector in the `i` th dimension.
- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: `dims.n[i]` >= 1 for i in 0, `rank-1`

`howmany_rank` Input parameter

`howmany_rank` is `int`

`howmany_rank` is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then `howmany_rank` needs only be 1.

Constraint: `howmany` >= 0

Note: If `howmany_rank=0`, then a single FFT transform is computed.

howmany_dims Input parameter

howmany_dims is const fftwh_iodim *

howmany_dims is an array of structures, dimension [rank], where the members of dims[i], for i in 0, rank-1, are defined to be:

- int n – the number of transforms to be computed of the i th dimension.of the howmany_rank-dimensional space mapped by the transforms to be computed.
- int is – the number of memory elements, of type fftwh_complex, between the start of transforms in the i th dimension. of the howmany_rank-dimensional space mapped by the transforms to be computed.
- int os – the stride between successive elements in the output vector in the i th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the howmany_dims.n[] entries. For example, if howmany_rank=1, then howmany_dims[0].n is the memory distance between successive transforms.

Constraint: howmany_dims.n[i] >= 1 for i in 0, howmany_rank-1

in Input parameter

in is fftwh_complex*

in is a pointer to a previously allocated array of fftwh_complex data. At fftwh_execute() time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

out Input parameter

out is fftwh_complex*

out is a pointer to a previously allocated array of fftwh_complex data. At fftwh_execute() time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

sign Input parameter

sign is int

sign is an integer denoting whether a forward or backward FFT transform should be performed

Constraint: sign = -1 (FFTW_FORWARD) for a forward transform,

sign = 1 (FFTW_BACKWARD) for a backwards transform

flags Input parameter

flags is unsigned

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in flags are:

- FFTW_ESTIMATE – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- FFTW_MEASURE – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- FFTW_PATIENT – Produce a plan, testing more options than FFTW_MEASURE. Input and output data might be overwritten.
- FFTW_EXHAUSTIVE – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.

- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwh_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.3.24 fftwh_plan_guru_split_dft

`fftwh_plan_guru_split_dft` is a guru FFTW interface call for planning an *n*-dimensional complex-to-complex FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner. Input and output data is stored in separate arrays for real and imaginary parts.

Syntax

C specification:

```
#include "fftw3.h"

fftwh_plan fftwh_plan_guru_split_dft(int rank, const fftwh_iodim *dims,
                                     int howmany_rank,
                                     const fftwh_iodim *howmany_dims,
                                     __fp16 *ri, __fp16 *ii, __fp16 *ro,
                                     __fp16 *io, int sign, unsigned flags);
```

Returns

This function returns an `fftwh_plan` object that can be used in subsequent *fftwh_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwh_execute_dft* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

`dims` Input parameter

`dims` is `const fftwh_iodim *`

`dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the length of the `i` th dimension.
- `int is` – the stride between successive elements in the input vector in the `i` th dimension.
- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: `dims.n[i]` >= 1 for i in 0, `rank-1`

`howmany_rank` Input parameter

`howmany_rank` is `int`

`howmany_rank` is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then `howmany_rank` needs only be 1.

Constraint: `howmany` >= 0

Note: If `howmany_rank=0`, then a single FFT transform is computed.

`howmany_dims` Input parameter

`howmany_dims` is `const fftwh_iodim *`

`howmany_dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the number of transforms to be computed of the `i` th dimension.of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `int is` – the number of memory elements, of type `fftwh_complex`, between the start of transforms in the `i` th dimension. of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the `howmany_dims.n[]` entries. For example, if `howmany_rank=1`, then `howmany_dims[0].n` is the memory distance between successive transforms.

Constraint: `howmany_dims.n[i]` >= 1 for i in 0, `howmany_rank-1`

`ri` Input parameter

`ri` is `__fp16 *`

`ri` is a pointer to a previously allocated array of `__fp16` data. At `fftwh_execute()` time this will have been filled with the real parts of the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`ii` Input parameter

`ii` is `__fp16 *`

`ii` is a pointer to a previously allocated array of `__fp16` data. At `fftwh_execute()` time this will have been filled with the imaginary parts of the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`ro` Input parameter

`ro` is `__fp16 *`

`ro` is a pointer to a previously allocated array of `__fp16` data. At `fftwh_execute()` time this will be filled with the real parts of the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

io Input parameter

io is __fp16 *

io is a pointer to a previously allocated array of __fp16 data. At fftwh_execute() time this will be filled with the imaginary parts of the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

sign Input parameter

sign is int

sign is an integer denoting whether a forward or backward FFT transform should be performed

Constraint: sign = -1 (FFTW_FORWARD) for a forward transform,

sign = 1 (FFTW_BACKWARD) for a backwards transform

flags Input parameter

flags is unsigned

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in flags are:

- FFTW_ESTIMATE – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- FFTW_MEASURE – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- FFTW_PATIENT – Produce a plan, testing more options than FFTW_MEASURE. Input and output data might be overwritten.
- FFTW_EXHAUSTIVE – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- FFTW_WISDOM_ONLY – Produce a plan only if the requested case has existing wisdom, otherwise return NULL.
- FFTW_PRESERVE_INPUT – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- FFTW_DESTROY_INPUT – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- FFTW_CONSERVE_MEMORY – Request that extra memory is not created. This might be ignored.
- FFTW_UNALIGNED – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by FFTW_MEASURE, FFTW_PATIENT and FFTW_EXHAUSTIVE will be limited when using *fftwh_set_timelimit*.
- If multiple options for planning rigor are included in flags, then the highest level of planning requested will be used.

5.3.25 fftwh_plan_guru64_dft

fftwh_plan_guru64_dft is a guru FFTW interface call for planning an *n*-dimensional complex-to-complex FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner.

Unlike *fftwh_plan_guru_dft*, `fftwh_plan_guru64_dft` allows selected parameters to be specified as 64-bit integers. For Arm Performance Libraries, this parameter size difference is true in both `armpl*_lp64` and `armpl_ilp64` variants. Therefore, the parameter size difference is independent of the integer size used in the rest of the library interface.

Syntax

C specification:

```
#include "fftw3.h"

fftwh_plan fftwh_plan_guru64_dft(int rank, const fftwh_iodim64 *dims,
                                 int howmany_rank,
                                 const fftwh_iodim64 *howmany_dims,
                                 fftwh_complex *in, fftwh_complex *out,
                                 int sign, unsigned flags);
```

Returns

This function returns an `fftwh_plan` object that can be used in subsequent *fftwh_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwh_execute_dft* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

`dims` Input parameter

`dims` is `const fftwh_iodim64 *`

`dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `ptrdiff_t n` – the length of the `i` th dimension.
- `ptrdiff_t is` – the stride between successive elements in the input vector in the `i` th dimension.
- `ptrdiff_t os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: `dims.n[i]` >= 1 for i in 0, `rank-1`

`howmany_rank` Input parameter

`howmany_rank` is `int`

`howmany_rank` is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then `howmany_rank` needs only be 1.

Constraint: `howmany` >= 0

Note: If `howmany_rank=0`, then a single FFT transform is computed.

howmany_dims Input parameter

howmany_dims is const fftwh_iodim64 *

howmany_dims is an array of structures, dimension [rank], where the members of dims[i], for i in 0, rank-1, are defined to be:

- ptrdiff_t n – the number of transforms to be computed of the i th dimension.of the howmany_rank-dimensional space mapped by the transforms to be computed.
- ptrdiff_t is – the number of memory elements, of type fftwh_complex, between the start of transforms in the i th dimension. of the howmany_rank-dimensional space mapped by the transforms to be computed.
- ptrdiff_t os – the stride between successive elements in the output vector in the i th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the howmany_dims.n[] entries. For example, if howmany_rank=1, then howmany_dims[0].n is the memory distance between successive transforms.

Constraint: howmany_dims.n[i] >= 1 for i in 0, howmany_rank-1

in Input parameter

in is fftwh_complex*

in is a pointer to a previously allocated array of fftwh_complex data. At fftwh_execute() time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

out Input parameter

out is fftwh_complex*

out is a pointer to a previously allocated array of fftwh_complex data. At fftwh_execute() time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

sign Input parameter

sign is int

sign is an integer denoting whether a forward or backward FFT transform should be performed

Constraint: sign = -1 (FFTW_FORWARD) for a forward transform,

sign = 1 (FFTW_BACKWARD) for a backwards transform

flags Input parameter

flags is unsigned

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in flags are:

- FFTW_ESTIMATE – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- FFTW_MEASURE – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- FFTW_PATIENT – Produce a plan, testing more options than FFTW_MEASURE. Input and output data might be overwritten.
- FFTW_EXHAUSTIVE – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.

- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwh_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.3.26 fftwh_plan_guru64_split_dft

`fftwh_plan_guru64_split_dft` is a guru FFTW interface call for planning an *n*-dimensional complex-to-complex FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner. Input and output data is stored in separate arrays for real and imaginary parts.

Unlike *fftwh_plan_guru_split_dft*, `fftwh_plan_guru64_split_dft` allows selected parameters to be specified as 64-bit integers. For Arm Performance Libraries, this parameter size difference is true in both `armpl*_lp64` and `armpl_ilp64` variants. Therefore, the parameter size difference is independent of the integer size used in the rest of the library interface.

Syntax

C specification:

```
#include "fftw3.h"

fftwh_plan fftwh_plan_guru_split_dft(int rank, const fftwh_iodim64 *dims,
                                     int howmany_rank,
                                     const fftwh_iodim64 *howmany_dims,
                                     __fp16 *ri, __fp16 *ii, __fp16 *ro,
                                     __fp16 *io, int sign, unsigned flags);
```

Returns

This function returns an `fftwh_plan` object that can be used in subsequent *fftwh_execute*`()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwh_execute_dft*`()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

dims Input parameter

dims is const fftwh_iodim64 *

dims is an array of structures, dimension [rank], where the members of dims[i], for i in 0, rank-1, are defined to be:

- ptrdiff_t n – the length of the i th dimension.
- ptrdiff_t is – the stride between successive elements in the input vector in the i th dimension.
- ptrdiff_t os – the stride between successive elements in the output vector in the i th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: dims.n[i] >= 1 for i in 0, rank-1

howmany_rank Input parameter

howmany_rank is int

howmany_rank is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then howmany_rank needs only be 1.

Constraint: howmany >= 0

Note: If howmany_rank=0, then a single FFT transform is computed.

howmany_dims Input parameter

howmany_dims is const fftwh_iodim64 *

howmany_dims is an array of structures, dimension [rank], where the members of dims[i], for i in 0, rank-1, are defined to be:

- ptrdiff_t n – the number of transforms to be computed of the i th dimension.of the howmany_rank-dimensional space mapped by the transforms to be computed.
- ptrdiff_t is – the number of memory elements, of type fftwh_complex, between the start of transforms in the i th dimension. of the howmany_rank-dimensional space mapped by the transforms to be computed.
- ptrdiff_t os – the stride between successive elements in the output vector in the i th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the howmany_dims.n[] entries. For example, if howmany_rank=1, then howmany_dims[0].n is the memory distance between successive transforms.

Constraint: howmany_dims.n[i] >= 1 for i in 0, howmany_rank-1

ri Input parameter

ri is __fp16 *

ri is a pointer to a previously allocated array of __fp16 data. At fftwh_execute() time this will have been filled with the real parts of the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

ii Input parameter

ri is __fp16 *

ri is a pointer to a previously allocated array of __fp16 data. At fftwh_execute() time this will have been filled with the imaginary parts of the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

ro Input parameter

ro is __fp16 *

ro is a pointer to a previously allocated array of __fp16 data. At fftwh_execute() time this will be filled with the real parts of the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

io Input parameter

io is __fp16 *

io is a pointer to a previously allocated array of __fp16 data. At fftwh_execute() time this will be filled with the imaginary parts of the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

sign Input parameter

sign is int

sign is an integer denoting whether a forward or backward FFT transform should be performed

Constraint: sign = -1 (FFTW_FORWARD) for a forward transform,

sign = 1 (FFTW_BACKWARD) for a backwards transform

flags Input parameter

flags is unsigned

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in flags are:

- FFTW_ESTIMATE – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- FFTW_MEASURE – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- FFTW_PATIENT – Produce a plan, testing more options than FFTW_MEASURE. Input and output data might be overwritten.
- FFTW_EXHAUSTIVE – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- FFTW_WISDOM_ONLY – Produce a plan only if the requested case has existing wisdom, otherwise return NULL.
- FFTW_PRESERVE_INPUT – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- FFTW_DESTROY_INPUT – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- FFTW_CONSERVE_MEMORY – Request that extra memory is not created. This might be ignored.
- FFTW_UNALIGNED – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by FFTW_MEASURE, FFTW_PATIENT and FFTW_EXHAUSTIVE will be limited when using *fftwh_set_timelimit*.
- If multiple options for planning rigor are included in flags, then the highest level of planning requested will be used.

5.3.27 fftwh_plan_many_dft

`fftwh_plan_many_dft` is an advanced FFTW interface call for planning an *n*-dimensional complex-to-complex FFT operation on `howmany` sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner.

For more information on the use of advanced planning calls see *How to choose which FFT routine you need*.

Syntax

C specification:

```
#include "fftw3.h"

fftwh_plan fftwh_plan_many_dft(int rank, const int *n, int howmany,
                               fftwh_complex *in, const int *inembed,
                               int istride, int idist, fftwh_complex *out,
                               const int *onembed, int ostride,
                               int odist, int sign, unsigned flags);
```

Returns

This function returns an `fftwh_plan` object that can be used in subsequent *fftwh_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwh_execute_dft* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of the FFT problem to be solved

Constraint: `rank` >= 1

`n` Input parameter

`n` is `const int*`

`n` is an array of dimension `[rank]`.

`n` contains the size of each dimension of the FFT to be solved

Constraint: `n[i]` >= 1 for i in 0, `rank-1`

`howmany` Input parameter

`howmany` is `int`

`howmany` is the number of multi-dimensional FFT transforms to be computed

Constraint: `howmany` >= 1

`in` Input parameter

`in` is `fftwh_complex*`

`in` is a pointer to a previously allocated array of `fftwh_complex` data. At `fftwh_execute()` time this will have been filled with the input sequence on which to perform the FFT.. The input data for set `k` starts at address `in+k*idist`.

Storage: Data in arrays will be stored in row-major order.

inembed Input parameter

inembed is const int

inembed is an array of dimension [rank] or NULL This array denotes the dimensionality of a potentially larger space of which the input array, in, dimensions given by n, is a sub array.

Constraint: Either inembed[i] >= n[i] for i in 0, rank-1 OR if inembed == NULL, then inembed is assumed to have the same dimensions as n

istride Input parameter

istride is int

istride is the distance between successive elements of the input data

idist Input parameter

idist is int

idist is the distance between the start of each of the howmany transforms in the input data

out Input parameter

out is fftwh_complex*

out is a pointer to a previously allocated array of fftwh_complex data. At fftwh_execute() time this will be filled with the resulting transform coefficients. The output data for set k starts at address out+k*odist.

Storage: Data in arrays will be stored in row-major order.

onembed Input parameter

onembed is const int

onembed is an array of dimension [rank] or NULL. This array denotes the dimensionality of a potentially larger space of which the input array, out, dimensions given by n, is a sub array.

Constraint: Either onembed[i] >= n[i] for i in 0, rank-1 OR if onembed == NULL, then onembed is assumed to have the same dimensions as n

ostride Input parameter

ostride is int

ostride is the distance between successive elements of the output data

odist Input parameter

odist is int

odist is the distance between the start of each of the howmany transforms in the output data

sign Input parameter

sign is int

sign is an integer denoting whether a forward or backward FFT transform should be performed

Constraint: sign = -1 (FFTW_FORWARD) for forward transform,

sign = 1 (FFTW_BACKWARD) for backwards transform

flags Input parameter

flags is unsigned

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in flags are:

- FFTW_ESTIMATE – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.

- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwh_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.4 FFT real to complex functions

5.4.1 fftw_plan_dft_r2c

`fftw_plan_dft_r2c` is a basic FFTW interface call for planning an *n*-dimensional real-to-complex FFT operation on a single, contiguous data sequence.

Note: `in` and `out` can be the same pointer for an "in-place" transform.

Syntax

C specification:

```
#include "fftw3.h"

fftw_plan fftw_plan_dft_r2c(int rank, const int *n, double *in,
                            fftw_complex *out, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_plan_dft_r2c(rank,n,in,out,flags) &
    bind(C, name='fftw_plan_dft_r2c')
  integer(C_INT), value :: rank
  integer(C_INT), dimension(*), intent(in) :: n
```

(continues on next page)

(continued from previous page)

```
  real(C_DOUBLE), dimension(*), intent(out) :: in
  complex(C_DOUBLE_COMPLEX), dimension(*), intent(out) :: out
  integer(C_INT), value :: flags
end function fftw_plan_dft_r2c
```

Returns

An `fftw_plan` object is returned that can be used in subsequent *fftw_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftw_execute_dft_r2c* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of the FFT problem to be solved

Constraint: rank >= 1

`n` Input parameter

`n` is a `const int*`.

`n` is an array of dimension `[rank]`.

`n` contains the size of each dimension of the FFT to be solved

Constraint: `n[i]` >=1 for i in 1, rank

`in` Input parameter

`in` is `double*`

`in` is a pointer to a previously allocated array of `double` data. At `fftw_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `fftw_complex*`

`out` is a pointer to a previously allocated array of `fftw_complex` data. At `fftw_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.

- FFTW_WISDOM_ONLY – Produce a plan only if the requested case has existing wisdom, otherwise return NULL.
- FFTW_PRESERVE_INPUT – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- FFTW_DESTROY_INPUT – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- FFTW_CONSERVE_MEMORY – Request that extra memory is not created. This might be ignored.
- FFTW_UNALIGNED – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by FFTW_MEASURE, FFTW_PATIENT and FFTW_EXHAUSTIVE will be limited when using *fftw_set_timelimit*.
- If multiple options for planning rigor are included in flags, then the highest level of planning requested will be used.

5.4.2 fftw_plan_dft_r2c_1d

fftw_plan_dft_r2c_1d is a basic FFTW interface call for planning a 1-d real-to-complex FFT operation on a single, contiguous data sequence.

Note: in and out can be the same pointer for an "in-place" transform.

Syntax

C specification:

```
#include "fftw3.h"

fftw_plan fftw_plan_dft_r2c_1d(int n0, double *in, fftw_complex *out,
                               unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_plan_dft_r2c_1d(n,in,out,flags) &
    bind(C, name='fftw_plan_dft_r2c_1d')
  integer(C_INT), value :: n
  real(C_DOUBLE), dimension(*), intent(out) :: in
  complex(C_DOUBLE_COMPLEX), dimension(*), intent(out) :: out
  integer(C_INT), value :: flags
end function fftw_plan_dft_r2c_1d
```

Returns

An fftw_plan object is returned that can be used in subsequent *fftw_execute*() calls on the previously supplied in and out arrays, or using different arrays via *fftw_execute_dft_r2c*().

Parameters

n0 Input parameter

`n0` is `int`

This is the length of the FFT sequence

Constraint: `n0` >= 1

in Input parameter

`in` is `double*`

`in` is a pointer to a previously allocated array of `double` data. At `fftw_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

out Input parameter

`out` is `fftw_complex*`

`out` is a pointer to a previously allocated array of `fftw_complex` data. At `fftw_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

flags Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftw_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.4.3 fftw_plan_dft_r2c_2d

`fftw_plan_dft_r2c_2d` is a basic FFTW interface call for planning a 2-d real-to-complex FFT operation on a single, contiguous data sequence.

Note: `in` and `out` can be the same pointer for an "in-place" transform.

Syntax

C specification:

```
#include "fftw3.h"

fftw_plan fftw_plan_dft_r2c_2d(int n0, int n1, double *in,
                               fftw_complex *out, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_plan_dft_r2c_2d(n0,n1,in,out,flags) &
    bind(C, name='fftw_plan_dft_r2c_2d')
  integer(C_INT), value :: n0
  integer(C_INT), value :: n1
  real(C_DOUBLE), dimension(*), intent(out) :: in
  complex(C_DOUBLE_COMPLEX), dimension(*), intent(out) :: out
  integer(C_INT), value :: flags
end function fftw_plan_dft_r2c_2d
```

Returns

An `fftw_plan` object is returned that can be used in subsequent *fftw_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftw_execute_dft_r2c* `()`.

Parameters

`n0` Input parameter

`n0` is `int`

This is the length of the first dimension of the FFT sequence

Constraint: `n0` >= 1

`n1` Input parameter

`n1` is `int`

This is the length of the second dimension of the FFT sequence

Constraint: `n1` >= 1

`in` Input parameter

`in` is `double*`

`in` is a pointer to a previously allocated array of `double` data. At `fftw_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

out Input parameter

`out` is fftw_complex

`out` is a pointer to a previously allocated array of `fftw_complex` data. At `fftw_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

flags Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftw_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.4.4 fftw_plan_dft_r2c_3d

`fftw_plan_dft_r2c_3d` is a basic FFTW interface call for planning a 3-d real-to-complex FFT operation on a single, contiguous data sequence.

Note: `in` and `out` can be the same pointer for an "in-place" transform.

Syntax

C specification:

```
#include "fftw3.h"

fftw_plan fftw_plan_dft_r2c_3d(int n0, int n1, int n2, double *in,
                               fftw_complex *out, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_plan_dft_r2c_3d(n0,n1,n2,in,out,flags) &
    bind(C, name='fftw_plan_dft_r2c_3d')
  integer(C_INT), value :: n0
  integer(C_INT), value :: n1
  integer(C_INT), value :: n2
  real(C_DOUBLE), dimension(*), intent(out) :: in
  complex(C_DOUBLE_COMPLEX), dimension(*), intent(out) :: out
  integer(C_INT), value :: flags
end function fftw_plan_dft_r2c_3d
```

Returns

An `fftw_plan` object is returned that can be used in subsequent *fftw_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftw_execute_dft_r2c* `()`.

Parameters

`n0` Input parameter

`n0` is `int`

This is the length of the first dimension of the FFT sequence

Constraint: `n0` >= 1

`n1` Input parameter

`n1` is `int`

This is the length of the second dimension of the FFT sequence

Constraint: `n1` >= 1

`n2` Input parameter

`n2` is `int`

This is the length of the third dimension of the FFT sequence

Constraint: `n2` >= 1

`in` Input parameter

`in` is double

`in` is a pointer to a previously allocated array of `double` data. At `fftw_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `fftw_complex*`

`out` is a pointer to a previously allocated array of `fftw_complex` data. At `fftw_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

flags Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftw_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.4.5 fftw_plan_guru_dft_r2c

`fftw_plan_guru_dft_r2c` is a guru FFTW interface call for planning an *n*-dimensional real-to-complex FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner.

Syntax

C specification:

```
#include "fftw3.h"

fftw_plan fftw_plan_guru_dft_r2c(int rank, const fftw_iodim *dims,
                                 int howmany_rank,
                                 const fftw_iodim *howmany_dims,
                                 double *in, fftw_complex *out,
                                 unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_plan_guru_dft_r2c(rank,dims,howmany_rank, &
    howmany_dims,in,out,flags) &
    bind(C, name='fftw_plan_guru_dft_r2c')
  integer(C_INT), value :: rank
  type(fftw_iodim), dimension(*), intent(in) :: dims
  integer(C_INT), value :: howmany_rank
  type(fftw_iodim), dimension(*), intent(in) :: howmany_dims
  real(C_DOUBLE), dimension(*), intent(out) :: in
  complex(C_DOUBLE_COMPLEX), dimension(*), intent(out) :: out
  integer(C_INT), value :: flags
end function fftw_plan_guru_dft_r2c
```

Returns

An `fftw_plan` object is returned that can be used in subsequent *fftw_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftw_execute_dft_r2c* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

`dims` Input parameter

`dims` is `const fftw_iodim *`

`dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the length of the `i` th dimension.
- `int is` – the stride between successive elements in the input vector in the `i` th dimension.
- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: `dims.n[i]` >= 1 for i in 0, `rank-1`

`howmany_rank` Input parameter

`howmany_rank` is `int`

`howmany_rank` is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then `howmany_rank` needs only be 1.

Constraint: `howmany` >= 0

Note: If `howmany_rank=0`, then a single FFT transform is computed.

`howmany_dims` Input parameter

`howmany_dims` is `const fftw_iodim *`

`howmany_dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the number of transforms to be computed of the `i` th dimension.of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `int is` – the number of memory elements, of type `fftw_complex`, between the start of transforms in the `i` th dimension. of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the `howmany_dims.n[]` entries. For example, if `howmany_rank=1`, then `howmany_dims[0].n` is the memory distance between successive transforms.

Constraint: `howmany_dims.n[i]` >= 1 for i in 0, `howmany_rank-1`

`in` Input parameter

`in` is `double*`

`in` is a pointer to a previously allocated array of `double` data. At `fftw_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `fftw_complex*`

`out` is a pointer to a previously allocated array of `fftw_complex` data. At `fftw_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftw_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.4.6 fftw_plan_guru_split_dft_r2c

`fftw_plan_guru_split_dft_r2c` is a guru FFTW interface call for planning an *n*-dimensional real-to-complex FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner. Input and output data is stored in separate arrays for real and imaginary parts.

Syntax

C specification:

```
#include "fftw3.h"

fftw_plan fftw_plan_guru_split_dft_r2c(int rank, const fftw_iodim *dims,
                                       int howmany_rank,
                                       const fftw_iodim *howmany_dims,
                                       double *in, double *ro,
                                       double *io, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_plan_guru_split_dft_r2c(rank,dims, &
    howmany_rank,howmany_dims,in,ro,io,flags) &
    bind(C, name='fftw_plan_guru_split_dft_r2c')
  integer(C_INT), value :: rank
  type(fftw_iodim), dimension(*), intent(in) :: dims
  integer(C_INT), value :: howmany_rank
  type(fftw_iodim), dimension(*), intent(in) :: howmany_dims
  real(C_DOUBLE), dimension(*), intent(out) :: in
  real(C_DOUBLE), dimension(*), intent(out) :: ro
  real(C_DOUBLE), dimension(*), intent(out) :: io
  integer(C_INT), value :: flags
end function fftw_plan_guru_split_dft_r2c
```

Returns

An `fftw_plan` object is returned that can be used in subsequent *fftw_execute*`()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftw_execute_dft_r2c*`()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

`dims` Input parameter

`dims` is `const fftw_iodim *`

`dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the length of the `i` th dimension.
- `int is` – the stride between successive elements in the input vector in the `i` th dimension.
- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: `dims.n[i]` >= 1 for i in 0, `rank-1`

howmany_rank Input parameter

`howmany_rank` is `int`

`howmany_rank` is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then `howmany_rank` needs only be 1.

Constraint: `howmany` >= 0

Note: If `howmany_rank=0`, then a single FFT transform is computed.

howmany_dims Input parameter

`howmany_dims` is `const fftw_iodim *`

`howmany_dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the number of transforms to be computed of the `i` th dimension.of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `int is` – the number of memory elements, of type `fftw_complex`, between the start of transforms in the `i` th dimension. of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the `howmany_dims.n[]` entries. For example, if `howmany_rank=1`, then `howmany_dims[0].n` is the memory distance between successive transforms.

Constraint: `howmany_dims.n[i]` >= 1 for i in 0, `howmany_rank-1`

in Input parameter

`in` is `double *`

`in` is a pointer to a previously allocated array of `double` data. At `fftw_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

ro Input parameter

`ro` is `double *`

`ro` is a pointer to a previously allocated array of `double` data. At `fftw_execute()` time this will be filled with the real parts of the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

io Input parameter

`io` is `double *`

`io` is a pointer to a previously allocated array of `double` data. At `fftw_execute()` time this will be filled with the imaginary parts of the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

flags Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftw_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.4.7 fftw_plan_guru64_dft_r2c

`fftw_plan_guru64_dft_r2c` is a guru FFTW interface call for planning an *n*-dimensional real-to-complex FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner.

Unlike *fftw_plan_guru_dft_r2c*, `fftw_plan_guru64_dft_r2c` allows selected parameters to be specified as 64-bit integers. For Arm Performance Libraries, this parameter size difference is true in both `armpl*_lp64` and `armpl_ilp64` variants. Therefore, the parameter size difference is independent of the integer size used in the rest of the library interface.

Syntax

C specification:

```
#include "fftw3.h"

fftw_plan fftw_plan_guru64_dft_r2c(int rank, const fftw_iodim64 *dims,
                                   int howmany_rank,
```

(continues on next page)

(continued from previous page)

```
                                    const fftw_iodim64 *howmany_dims,
                                    double *in, fftw_complex *out,
                                    unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_plan_guru64_dft_r2c(rank,dims,howmany_rank, &
    howmany_dims,in,out,flags) &
    bind(C, name='fftw_plan_guru64_dft_r2c')
  integer(C_INT), value :: rank
  type(fftw_iodim64), dimension(*), intent(in) :: dims
  integer(C_INT), value :: howmany_rank
  type(fftw_iodim64), dimension(*), intent(in) :: howmany_dims
  real(C_DOUBLE), dimension(*), intent(out) :: in
  complex(C_DOUBLE_COMPLEX), dimension(*), intent(out) :: out
  integer(C_INT), value :: flags
end function fftw_plan_guru64_dft_r2c
```

Returns

An `fftw_plan` object is returned that can be used in subsequent *fftw_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftw_execute_dft_r2c* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

`dims` Input parameter

`dims` is `const fftw_iodim64 *`

`dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `ptrdiff_t n` – the length of the `i` th dimension.
- `ptrdiff_t is` – the stride between successive elements in the input vector in the `i` th dimension.
- `ptrdiff_t os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: `dims.n[i]` >= 1 for i in 0, `rank-1`

`howmany_rank` Input parameter

`howmany_rank` is `int`

`howmany_rank` is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then `howmany_rank` needs only be 1.

Constraint: `howmany` >= 0

Note: If `howmany_rank=0`, then a single FFT transform is computed.

howmany_dims Input parameter

howmany_dims is const fftw_iodim64 *

howmany_dims is an array of structures, dimension [rank], where the members of dims[i], for i in 0, rank-1, are defined to be:

- ptrdiff_t n – the number of transforms to be computed of the i th dimension.of the howmany_rank-dimensional space mapped by the transforms to be computed.
- ptrdiff_t is – the number of memory elements, of type fftw_complex, between the start of transforms in the i th dimension. of the howmany_rank-dimensional space mapped by the transforms to be computed.
- ptrdiff_t os – the stride between successive elements in the output vector in the i th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the howmany_dims.n[] entries. For example, if howmany_rank=1, then howmany_dims[0].n is the memory distance between successive transforms.

Constraint: howmany_dims.n[i] >= 1 for i in 0, howmany_rank-1

in Input parameter

in is double*

in is a pointer to a previously allocated array of double data. At fftw_execute() time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

out Input parameter

out is fftw_complex*

out is a pointer to a previously allocated array of fftw_complex data. At fftw_execute() time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

flags Input parameter

flags is unsigned

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in flags are:

- FFTW_ESTIMATE – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- FFTW_MEASURE – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- FFTW_PATIENT – Produce a plan, testing more options than FFTW_MEASURE. Input and output data might be overwritten.
- FFTW_EXHAUSTIVE – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- FFTW_WISDOM_ONLY – Produce a plan only if the requested case has existing wisdom, otherwise return NULL.
- FFTW_PRESERVE_INPUT – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- FFTW_DESTROY_INPUT – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.

- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftw_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.4.8 fftw_plan_guru64_split_dft_r2c

`fftw_plan_guru64_split_dft_r2c` is a guru FFTW interface call for planning an *n*-dimensional real-to-complex FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner. Input and output data is stored in separate arrays for real and imaginary parts.

Unlike *fftw_plan_guru_split_dft_r2c*, `fftw_plan_guru64_split_dft_r2c` allows selected parameters to be specified as 64-bit integers. For Arm Performance Libraries, this parameter size difference is true in both `armpl*_lp64` and `armpl_ilp64` variants. Therefore, the parameter size difference is independent of the integer size used in the rest of the library interface.

Syntax

C specification:

```
#include "fftw3.h"

fftw_plan fftw_plan_guru_split_dft_r2c(int rank, const fftw_iodim64 *dims,
                                       int howmany_rank,
                                       const fftw_iodim64 *howmany_dims,
                                       double *in, double *ro,
                                       double *io, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_plan_guru64_split_dft_r2c(rank,dims,howmany_rank,howmany_
↪dims,in,ro,io,flags) &
                     bind(C, name='fftw_plan_guru64_split_dft_r2c')
  integer(C_INT), value :: rank
  type(fftw_iodim64), dimension(*), intent(in) :: dims
  integer(C_INT), value :: howmany_rank
  type(fftw_iodim64), dimension(*), intent(in) :: howmany_dims
  real(C_DOUBLE), dimension(*), intent(out) :: in
  real(C_DOUBLE), dimension(*), intent(out) :: ro
  real(C_DOUBLE), dimension(*), intent(out) :: io
  integer(C_INT), value :: flags
end function fftw_plan_guru64_split_dft_r2c
```

Returns

An `fftw_plan` object is returned that can be used in subsequent *fftw_execute*`()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftw_execute_dft_r2c*`()`.

Parameters

rank Input parameter

rank is int

The number of dimensions of each individual FFT problem to be solved.

Constraint: rank >= 1

dims Input parameter

dims is const fftw_iodim64 *

dims is an array of structures, dimension [rank], where the members of dims[i], for i in 0, rank-1, are defined to be:

- ptrdiff_t n – the length of the i th dimension.
- ptrdiff_t is – the stride between successive elements in the input vector in the i th dimension.
- ptrdiff_t os – the stride between successive elements in the output vector in the i th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: dims.n[i] >= 1 for i in 0, rank-1

howmany_rank Input parameter

howmany_rank is int

howmany_rank is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then howmany_rank needs only be 1.

Constraint: howmany >= 0

Note: If howmany_rank=0, then a single FFT transform is computed.

howmany_dims Input parameter

howmany_dims is const fftw_iodim64 *

howmany_dims is an array of structures, dimension [rank], where the members of dims[i], for i in 0, rank-1, are defined to be:

- ptrdiff_t n – the number of transforms to be computed of the i th dimension.of the howmany_rank-dimensional space mapped by the transforms to be computed.
- ptrdiff_t is – the number of memory elements, of type fftw_complex, between the start of transforms in the i th dimension. of the howmany_rank-dimensional space mapped by the transforms to be computed.
- ptrdiff_t os – the stride between successive elements in the output vector in the i th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the howmany_dims.n[] entries. For example, if howmany_rank=1, then howmany_dims[0].n is the memory distance between successive transforms.

Constraint: howmany_dims.n[i] >= 1 for i in 0, howmany_rank-1

in Input parameter

in is double *

in is a pointer to a previously allocated array of double data. At fftw_execute() time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

ro Input parameter

ro is double *

ro is a pointer to a previously allocated array of double data. At fftw_execute() time this will be filled with the real parts of the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

io Input parameter

io is double *

io is a pointer to a previously allocated array of double data. At fftw_execute() time this will be filled with the imaginary parts of the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

flags Input parameter

flags is unsigned

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in flags are:

- FFTW_ESTIMATE – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- FFTW_MEASURE – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- FFTW_PATIENT – Produce a plan, testing more options than FFTW_MEASURE. Input and output data might be overwritten.
- FFTW_EXHAUSTIVE – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- FFTW_WISDOM_ONLY – Produce a plan only if the requested case has existing wisdom, otherwise return NULL.
- FFTW_PRESERVE_INPUT – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- FFTW_DESTROY_INPUT – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- FFTW_CONSERVE_MEMORY – Request that extra memory is not created. This might be ignored.
- FFTW_UNALIGNED – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by FFTW_MEASURE, FFTW_PATIENT and FFTW_EXHAUSTIVE will be limited when using *fftw_set_timelimit*.
- If multiple options for planning rigor are included in flags, then the highest level of planning requested will be used.

5.4.9 fftw_plan_many_dft_r2c

fftw_plan_many_dft_r2c is an advanced FFTW interface call for planning an *n*-dimensional real-to-complex FFT operation on howmany sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner.

For more information on the use of advanced planning calls see *How to choose which FFT routine you need*.

Syntax

C specification:

```
#include "fftw3.h"

fftw_plan fftw_plan_many_dft_r2c(int rank, const int *n, int howmany,
                                 double *in, const int *inembed,
                                 int istride, int idist, fftw_complex *out,
                                 const int *onembed, int ostride, int odist,
                                 unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_plan_many_dft_r2c(rank,n,howmany,in,inembed, &
    istride,idist,out,onembed,ostride,odist,flags) &
    bind(C, name='fftw_plan_many_dft_r2c')
  integer(C_INT), value :: rank
  integer(C_INT), dimension(*), intent(in) :: n
  integer(C_INT), value :: howmany
  real(C_DOUBLE), dimension(*), intent(out) :: in
  integer(C_INT), dimension(*), intent(in) :: inembed
  integer(C_INT), value :: istride
  integer(C_INT), value :: idist
  complex(C_DOUBLE_COMPLEX), dimension(*), intent(out) :: out
  integer(C_INT), dimension(*), intent(in) :: onembed
  integer(C_INT), value :: ostride
  integer(C_INT), value :: odist
  integer(C_INT), value :: flags
end function fftw_plan_many_dft_r2c
```

Returns

An `fftw_plan` object is returned that can be used in subsequent *fftw_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftw_execute_dft_r2c* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of the FFT problem to be solved

Constraint: rank >= 1

`n` Input parameter

`n` is `const int`

`n` is an array of dimension `[rank]`.

`n` contains the size of each dimension of the FFT to be solved

Constraint: `n[i]` >= 1 for i in 1, rank

`howmany` Input parameter

`howmany` is `int`

`howmany` is the number of multi-dimensional FFT transforms to be computed

Constraint: howmany >= 1

in Input parameter

in is double*

in is a pointer to a previously allocated array of double data. At fftw_execute() time this will have been filled with the input sequence on which to perform the FFT. The input data for set k starts at address in+k*idist.

Storage: Data in arrays will be stored in row-major order.

inembed Input parameter

inembed is const int

inembed is an array of dimension [rank] or NULL. This array denotes the dimensionality of a potentially larger space of which the input array, in, dimensions given by n, is a sub array.

Constraint: Either inembed[i] >= n[i] for i in 0, rank-1 OR if inembed == NULL, then inembed is assumed to have the same dimensions as n

istride Input parameter

istride is int

istride is the distance between successive elements of the input data

idist Input parameter

idist is int

idist is the distance between the start of each of the howmany transforms in the input data

out Input parameter

out is fftw_complex*

out is a pointer to a previously allocated array of fftw_complex data. At fftw_execute() time this will be filled with the resulting transform coefficients The output data for set k starts at address out+k*odist.

Storage: Data in arrays will be stored in row-major order.

onembed Input parameter

onembed is const int

onembed is an array of dimension [rank] or NULL. This array denotes the dimensionality of a potentially larger space of which the input array, out, dimensions given by n, is a sub array.

Constraint: Either onembed[i] >= n[i] for i in 0, rank-1 OR if onembed == NULL, then onembed is assumed to have the same dimensions as n

ostride Input parameter

ostride is int

ostride is the distance between successive elements of the output data

odist Input parameter

odist is int

odist is the distance between the start of each of the howmany transforms in the output data

flags Input parameter

flags is unsigned

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in flags are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftw_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.4.10 fftwf_plan_dft_r2c

`fftwf_plan_dft_r2c` is a basic FFTW interface call for planning an *n*-dimensional real-to-complex FFT operation on a single, contiguous data sequence.

Note: `in` and `out` can be the same pointer for an "in-place" transform.

Syntax

C specification:

```
#include "fftw3.h"

fftwf_plan fftwf_plan_dft_r2c(int rank, const int *n, float *in,
                              fftwf_complex *out, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_plan_dft_r2c(rank,n,in,out,flags) &
    bind(C, name='fftwf_plan_dft_r2c')
  integer(C_INT), value :: rank
  integer(C_INT), dimension(*), intent(in) :: n
  real(C_FLOAT), dimension(*), intent(out) :: in
```

(continues on next page)

(continued from previous page)

```
  complex(C_FLOAT_COMPLEX), dimension(*), intent(out) :: out
  integer(C_INT), value :: flags
end function fftwf_plan_dft_r2c
```

Returns

This function returns an `fftwf_plan` object that can be used in subsequent *fftwf_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwf_execute_dft_r2c* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of the FFT problem to be solved

Constraint: rank >= 1

`n` Input parameter

`n` is a `const int*`.

`n` is an array of dimension `[rank]`.

`n` contains the size of each dimension of the FFT to be solved

Constraint: `n[i]` >=1 for i in 1, rank

`in` Input parameter

`in` is `float`

`in` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `fftwf_complex*`

`out` is a pointer to a previously allocated array of `fftwf_complex` data. At `fftwf_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.

- FFTW_WISDOM_ONLY – Produce a plan only if the requested case has existing wisdom, otherwise return NULL.
- FFTW_PRESERVE_INPUT – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- FFTW_DESTROY_INPUT – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- FFTW_CONSERVE_MEMORY – Request that extra memory is not created. This might be ignored.
- FFTW_UNALIGNED – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by FFTW_MEASURE, FFTW_PATIENT and FFTW_EXHAUSTIVE will be limited when using *fftwf_set_timelimit*.
- If multiple options for planning rigor are included in flags, then the highest level of planning requested will be used.

5.4.11 fftwf_plan_dft_r2c_1d

fftwf_plan_dft_r2c_1d is a basic FFTW interface call for planning a 1-d real-to-complex FFT operation on a single, contiguous data sequence.

Note: in and out can be the same pointer for an "in-place" transform.

Syntax

C specification:

```
#include "fftw3.h"

fftwf_plan fftwf_plan_dft_r2c_1d(int n0, float *in, fftwf_complex *out,
                                 unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_plan_dft_r2c_1d(n,in,out,flags) &
    bind(C, name='fftwf_plan_dft_r2c_1d')
  integer(C_INT), value :: n
  real(C_FLOAT), dimension(*), intent(out) :: in
  complex(C_FLOAT_COMPLEX), dimension(*), intent(out) :: out
  integer(C_INT), value :: flags
end function fftwf_plan_dft_r2c_1d
```

Returns

This function returns an fftwf_plan object that can be used in subsequent *fftwf_execute*() calls on the previously supplied in and out arrays, or using different arrays via *fftwf_execute_dft_r2c*().

Parameters

n0 Input parameter

n0 is int

This is the length of the FFT sequence

Constraint: n0 >= 1

in Input parameter

in is float

in is a pointer to a previously allocated array of float data. At fftwf_execute() time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

out Input parameter

out is fftwf_complex*

out is a pointer to a previously allocated array of fftwf_complex data. At fftwf_execute() time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

flags Input parameter

flags is unsigned

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in flags are:

- FFTW_ESTIMATE – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- FFTW_MEASURE – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- FFTW_PATIENT – Produce a plan, testing more options than FFTW_MEASURE. Input and output data might be overwritten.
- FFTW_EXHAUSTIVE – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- FFTW_WISDOM_ONLY – Produce a plan only if the requested case has existing wisdom, otherwise return NULL.
- FFTW_PRESERVE_INPUT – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- FFTW_DESTROY_INPUT – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- FFTW_CONSERVE_MEMORY – Request that extra memory is not created. This might be ignored.
- FFTW_UNALIGNED – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by FFTW_MEASURE, FFTW_PATIENT and FFTW_EXHAUSTIVE will be limited when using *fftwf_set_timelimit*.
- If multiple options for planning rigor are included in flags, then the highest level of planning requested will be used.

5.4.12 fftwf_plan_dft_r2c_2d

`fftwf_plan_dft_r2c_2d` is a basic FFTW interface call for planning a 2-d real-to-complex FFT operation on a single, contiguous data sequence.

Note: `in` and `out` can be the same pointer for an "in-place" transform.

Syntax

C specification:

```
#include "fftw3.h"

fftwf_plan fftwf_plan_dft_r2c_2d(int n0, int n1, float *in,
                                 fftwf_complex *out, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_plan_dft_r2c_2d(n0,n1,in,out,flags) &
    bind(C, name='fftwf_plan_dft_r2c_2d')
  integer(C_INT), value :: n0
  integer(C_INT), value :: n1
  real(C_FLOAT), dimension(*), intent(out) :: in
  complex(C_FLOAT_COMPLEX), dimension(*), intent(out) :: out
  integer(C_INT), value :: flags
end function fftwf_plan_dft_r2c_2d
```

Returns

This function returns an `fftwf_plan` object that can be used in subsequent *fftwf_execute*`()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwf_execute_dft_r2c*`()`.

Parameters

n0 Input parameter

`n0` is `int`

This is the length of the first dimension of the FFT sequence

Constraint: `n0` >= 1

n1 Input parameter

`n1` is `int`

This is the length of the second dimension of the FFT sequence

Constraint: `n1` >= 1

in Input parameter

`in` is `float`

`in` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

out Input parameter

`out` is fftwf_complex

`out` is a pointer to a previously allocated array of `fftwf_complex` data. At `fftwf_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

flags Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwf_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.4.13 fftwf_plan_dft_r2c_3d

`fftwf_plan_dft_r2c_3d` is a basic FFTW interface call for planning a 3-d real-to-complex FFT operation on a single, contiguous data sequence.

Note: `in` and `out` can be the same pointer for an "in-place" transform.

Syntax

C specification:

```
#include "fftw3.h"

fftwf_plan fftwf_plan_dft_r2c_3d(int n0, int n1, int n2, float *in,
                                 fftwf_complex *out, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_plan_dft_r2c_3d(n0,n1,n2,in,out,flags) &
    bind(C, name='fftwf_plan_dft_r2c_3d')
  integer(C_INT), value :: n0
  integer(C_INT), value :: n1
  integer(C_INT), value :: n2
  real(C_FLOAT), dimension(*), intent(out) :: in
  complex(C_FLOAT_COMPLEX), dimension(*), intent(out) :: out
  integer(C_INT), value :: flags
end function fftwf_plan_dft_r2c_3d
```

Returns

This function returns an `fftwf_plan` object that can be used in subsequent *fftwf_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwf_execute_dft_r2c* `()`.

Parameters

`n0` Input parameter

`n0` is `int`

This is the length of the first dimension of the FFT sequence

Constraint: `n0` >= 1

`n1` Input parameter

`n1` is `int`

This is the length of the second dimension of the FFT sequence

Constraint: `n1` >= 1

`n2` Input parameter

`n2` is `int`

This is the length of the third dimension of the FFT sequence

Constraint: `n2` >= 1

`in` Input parameter

`in` is float

`in` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `fftwf_complex*`

`out` is a pointer to a previously allocated array of `fftwf_complex` data. At `fftwf_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

flags Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwf_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.4.14 fftwf_plan_guru_dft_r2c

`fftwf_plan_guru_dft_r2c` is a guru FFTW interface call for planning an *n*-dimensional real-to-complex FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner.

Syntax

C specification:

```
#include "fftw3.h"

fftwf_plan fftwf_plan_guru_dft_r2c(int rank, const fftwf_iodim *dims,
                                   int howmany_rank,
                                   const fftwf_iodim *howmany_dims,
                                   float *in, fftwf_complex *out,
                                   unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_plan_guru_dft_r2c(rank,dims,howmany_rank, &
    howmany_dims,in,out,flags) &
    bind(C, name='fftwf_plan_guru_dft_r2c')
  integer(C_INT), value :: rank
  type(fftwf_iodim), dimension(*), intent(in) :: dims
  integer(C_INT), value :: howmany_rank
  type(fftwf_iodim), dimension(*), intent(in) :: howmany_dims
  real(C_FLOAT), dimension(*), intent(out) :: in
  complex(C_FLOAT_COMPLEX), dimension(*), intent(out) :: out
  integer(C_INT), value :: flags
end function fftwf_plan_guru_dft_r2c
```

Returns

This function returns an `fftwf_plan` object that can be used in subsequent *fftwf_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwf_execute_dft_r2c* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

`dims` Input parameter

`dims` is `const fftwf_iodim *`

`dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the length of the `i` th dimension.
- `int is` – the stride between successive elements in the input vector in the `i` th dimension.
- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: `dims.n[i]` >= 1 for i in 0, `rank-1`

`howmany_rank` Input parameter

`howmany_rank` is `int`

`howmany_rank` is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then `howmany_rank` needs only be 1.

Constraint: `howmany` >= 0

Note: If `howmany_rank=0`, then a single FFT transform is computed.

`howmany_dims` Input parameter

`howmany_dims` is `const fftwf_iodim *`

`howmany_dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the number of transforms to be computed of the `i` th dimension.of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `int is` – the number of memory elements, of type `fftwf_complex`, between the start of transforms in the `i` th dimension. of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the `howmany_dims.n[]` entries. For example, if `howmany_rank=1`, then `howmany_dims[0].n` is the memory distance between successive transforms.

Constraint: `howmany_dims.n[i]` >= 1 for i in 0, `howmany_rank-1`

`in` Input parameter

`in` is `float`

`in` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `fftwf_complex*`

`out` is a pointer to a previously allocated array of `fftwf_complex` data. At `fftwf_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwf_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.4.15 fftwf_plan_guru_split_dft_r2c

`fftwf_plan_guru_split_dft_r2c` is a guru FFTW interface call for planning an *n*-dimensional real-to-complex FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner. Input and output data is stored in separate arrays for real and imaginary parts.

Syntax

C specification:

```
#include "fftw3.h"

fftwf_plan fftwf_plan_guru_split_dft_r2c(int rank, const fftwf_iodim *dims,
                                         int howmany_rank,
                                         const fftwf_iodim *howmany_dims,
                                         float *in, float *ro,
                                         float *io, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_plan_guru_split_dft_r2c(rank,dims, &
    howmany_rank,howmany_dims,in,ro,io,flags) &
    bind(C, name='fftwf_plan_guru_split_dft_r2c')
  integer(C_INT), value :: rank
  type(fftwf_iodim), dimension(*), intent(in) :: dims
  integer(C_INT), value :: howmany_rank
  type(fftwf_iodim), dimension(*), intent(in) :: howmany_dims
  real(C_FLOAT), dimension(*), intent(out) :: in
  real(C_FLOAT), dimension(*), intent(out) :: ro
  real(C_FLOAT), dimension(*), intent(out) :: io
  integer(C_INT), value :: flags
end function fftwf_plan_guru_split_dft_r2c
```

Returns

This function returns an `fftwf_plan` object that can be used in subsequent *fftwf_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwf_execute_dft_r2c* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

`dims` Input parameter

`dims` is `const fftwf_iodim *`

`dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the length of the `i` th dimension.
- `int is` – the stride between successive elements in the input vector in the `i` th dimension.
- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: `dims.n[i]` >= 1 for i in 0, `rank-1`

howmany_rank Input parameter

`howmany_rank` is `int`

`howmany_rank` is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then `howmany_rank` needs only be 1.

Constraint: `howmany` >= 0

Note: If `howmany_rank=0`, then a single FFT transform is computed.

howmany_dims Input parameter

`howmany_dims` is `const fftwf_iodim *`

`howmany_dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the number of transforms to be computed of the `i` th dimension.of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `int is` – the number of memory elements, of type `fftwf_complex`, between the start of transforms in the `i` th dimension. of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the `howmany_dims.n[]` entries. For example, if `howmany_rank=1`, then `howmany_dims[0].n` is the memory distance between successive transforms.

Constraint: `howmany_dims.n[i]` >= 1 for i in 0, `howmany_rank-1`

in Input parameter

`in` is `float *`

`in` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

ro Input parameter

`ro` is `float *`

`ro` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will be filled with the real parts of the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

io Input parameter

`io` is `float *`

`io` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will be filled with the imaginary parts of the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

flags Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwf_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.4.16 fftwf_plan_guru64_dft_r2c

`fftwf_plan_guru64_dft_r2c` is a guru FFTW interface call for planning an *n*-dimensional real-to-complex FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner.

Unlike *fftwf_plan_guru_dft_r2c*, `fftwf_plan_guru64_dft_r2c` allows selected parameters to be specified as 64-bit integers. For Arm Performance Libraries, this parameter size difference is true in both `armpl*_lp64` and `armpl_ilp64` variants. Therefore, the parameter size difference is independent of the integer size used in the rest of the library interface.

Syntax

C specification:

```
#include "fftw3.h"

fftwf_plan fftwf_plan_guru64_dft_r2c(int rank, const fftwf_iodim64 *dims,
                                     int howmany_rank,
```

(continues on next page)

(continued from previous page)

```
                                    const fftwf_iodim64 *howmany_dims,
                                    float *in, fftwf_complex *out,
                                    unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_plan_guru64_dft_r2c(rank,dims,howmany_rank, &
    howmany_dims,in,out,flags) &
    bind(C, name='fftwf_plan_guru64_dft_r2c')
  integer(C_INT), value :: rank
  type(fftwf_iodim64), dimension(*), intent(in) :: dims
  integer(C_INT), value :: howmany_rank
  type(fftwf_iodim64), dimension(*), intent(in) :: howmany_dims
  real(C_FLOAT), dimension(*), intent(out) :: in
  complex(C_FLOAT_COMPLEX), dimension(*), intent(out) :: out
  integer(C_INT), value :: flags
end function fftwf_plan_guru64_dft_r2c
```

Returns

This function returns an `fftwf_plan` object that can be used in subsequent *fftwf_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwf_execute_dft_r2c* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

`dims` Input parameter

`dims` is `const fftwf_iodim64 *`

`dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `ptrdiff_t n` – the length of the `i` th dimension.
- `ptrdiff_t is` – the stride between successive elements in the input vector in the `i` th dimension.
- `ptrdiff_t os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: `dims.n[i]` >= 1 for i in 0, `rank-1`

`howmany_rank` Input parameter

`howmany_rank` is `int`

`howmany_rank` is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then `howmany_rank` needs only be 1.

Constraint: `howmany` >= 0

Note: If `howmany_rank=0`, then a single FFT transform is computed.

howmany_dims Input parameter

howmany_dims is const fftwf_iodim64 *

howmany_dims is an array of structures, dimension [rank], where the members of dims[i], for i in 0, rank-1, are defined to be:

- ptrdiff_t n – the number of transforms to be computed of the i th dimension.of the howmany_rank-dimensional space mapped by the transforms to be computed.
- ptrdiff_t is – the number of memory elements, of type fftwf_complex, between the start of transforms in the i th dimension. of the howmany_rank-dimensional space mapped by the transforms to be computed.
- ptrdiff_t os – the stride between successive elements in the output vector in the i th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the howmany_dims.n[] entries. For example, if howmany_rank=1, then howmany_dims[0].n is the memory distance between successive transforms.

Constraint: howmany_dims.n[i] >= 1 for i in 0, howmany_rank-1

in Input parameter

in is float

in is a pointer to a previously allocated array of float data. At fftwf_execute() time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

out Input parameter

out is fftwf_complex*

out is a pointer to a previously allocated array of fftwf_complex data. At fftwf_execute() time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

flags Input parameter

flags is unsigned

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in flags are:

- FFTW_ESTIMATE – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- FFTW_MEASURE – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- FFTW_PATIENT – Produce a plan, testing more options than FFTW_MEASURE. Input and output data might be overwritten.
- FFTW_EXHAUSTIVE – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- FFTW_WISDOM_ONLY – Produce a plan only if the requested case has existing wisdom, otherwise return NULL.
- FFTW_PRESERVE_INPUT – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- FFTW_DESTROY_INPUT – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.

- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwf_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.4.17 fftwf_plan_guru64_split_dft_r2c

`fftwf_plan_guru64_split_dft_r2c` is a guru FFTW interface call for planning an *n*-dimensional real-to-complex FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner. Input and output data is stored in separate arrays for real and imaginary parts.

Unlike *fftwf_plan_guru_split_dft_r2c*, `fftwf_plan_guru64_split_dft_r2c` allows selected parameters to be specified as 64-bit integers. For Arm Performance Libraries, this parameter size difference is true in both `armpl*_lp64` and `armpl_ilp64` variants. Therefore, the parameter size difference is independent of the integer size used in the rest of the library interface.

Syntax

C specification:

```
#include "fftw3.h"

fftwf_plan fftwf_plan_guru_split_dft_r2c(int rank, const fftwf_iodim64 *dims,
                                         int howmany_rank,
                                         const fftwf_iodim64 *howmany_dims,
                                         float *in, float *ro,
                                         float *io, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_plan_guru64_split_dft_r2c(rank,dims, &
    howmany_rank,howmany_dims,in,ro,io,flags) &
    bind(C, name='fftwf_plan_guru64_split_dft_r2c')
  integer(C_INT), value :: rank
  type(fftwf_iodim64), dimension(*), intent(in) :: dims
  integer(C_INT), value :: howmany_rank
  type(fftwf_iodim64), dimension(*), intent(in) :: howmany_dims
  real(C_FLOAT), dimension(*), intent(out) :: in
  real(C_FLOAT), dimension(*), intent(out) :: ro
  real(C_FLOAT), dimension(*), intent(out) :: io
  integer(C_INT), value :: flags
end function fftwf_plan_guru64_split_dft_r2c
```

Returns

This function returns an `fftwf_plan` object that can be used in subsequent *fftwf_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwf_execute_dft_r2c* `()`.

Parameters

rank Input parameter

rank is int

The number of dimensions of each individual FFT problem to be solved.

Constraint: rank >= 1

dims Input parameter

dims is const fftwf_iodim64 *

dims is an array of structures, dimension [rank], where the members of dims[i], for i in 0, rank-1, are defined to be:

- ptrdiff_t n – the length of the i th dimension.
- ptrdiff_t is – the stride between successive elements in the input vector in the i th dimension.
- ptrdiff_t os – the stride between successive elements in the output vector in the i th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: dims.n[i] >= 1 for i in 0, rank-1

howmany_rank Input parameter

howmany_rank is int

howmany_rank is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then howmany_rank needs only be 1.

Constraint: howmany >= 0

Note: If howmany_rank=0, then a single FFT transform is computed.

howmany_dims Input parameter

howmany_dims is const fftwf_iodim64 *

howmany_dims is an array of structures, dimension [rank], where the members of dims[i], for i in 0, rank-1, are defined to be:

- ptrdiff_t n – the number of transforms to be computed of the i th dimension.of the howmany_rank-dimensional space mapped by the transforms to be computed.
- ptrdiff_t is – the number of memory elements, of type fftwf_complex, between the start of transforms in the i th dimension. of the howmany_rank-dimensional space mapped by the transforms to be computed.
- ptrdiff_t os – the stride between successive elements in the output vector in the i th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the howmany_dims.n[] entries. For example, if howmany_rank=1, then howmany_dims[0].n is the memory distance between successive transforms.

Constraint: howmany_dims.n[i] >= 1 for i in 0, howmany_rank-1

in Input parameter

in is float *

in is a pointer to a previously allocated array of float data. At fftwf_execute() time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

ro Input parameter

ro is float *

ro is a pointer to a previously allocated array of float data. At fftwf_execute() time this will be filled with the real parts of the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

io Input parameter

io is float *

io is a pointer to a previously allocated array of float data. At fftwf_execute() time this will be filled with the imaginary parts of the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

flags Input parameter

flags is unsigned

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in flags are:

- FFTW_ESTIMATE – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- FFTW_MEASURE – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- FFTW_PATIENT – Produce a plan, testing more options than FFTW_MEASURE. Input and output data might be overwritten.
- FFTW_EXHAUSTIVE – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- FFTW_WISDOM_ONLY – Produce a plan only if the requested case has existing wisdom, otherwise return NULL.
- FFTW_PRESERVE_INPUT – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- FFTW_DESTROY_INPUT – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- FFTW_CONSERVE_MEMORY – Request that extra memory is not created. This might be ignored.
- FFTW_UNALIGNED – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by FFTW_MEASURE, FFTW_PATIENT and FFTW_EXHAUSTIVE will be limited when using *fftwf_set_timelimit*.
- If multiple options for planning rigor are included in flags, then the highest level of planning requested will be used.

5.4.18 fftwf_plan_many_dft_r2c

fftwf_plan_many_dft_r2c is an advanced FFTW interface call for planning an *n*-dimensional real-to-complex FFT operation on howmany sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner.

For more information on the use of advanced planning calls see *How to choose which FFT routine you need*.

Syntax

C specification:

```
#include "fftw3.h"

fftwf_plan fftwf_plan_many_dft_r2c(int rank, const int *n, int howmany,
                                   float *in, const int *inembed,
                                   int istride, int idist, fftwf_complex *out,
                                   const int *onembed, int ostride, int odist,
                                   unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_plan_many_dft_r2c(rank,n,howmany,in,inembed, &
    istride,idist,out,onembed,ostride,odist,flags) &
    bind(C, name='fftwf_plan_many_dft_r2c')
  integer(C_INT), value :: rank
  integer(C_INT), dimension(*), intent(in) :: n
  integer(C_INT), value :: howmany
  real(C_FLOAT), dimension(*), intent(out) :: in
  integer(C_INT), dimension(*), intent(in) :: inembed
  integer(C_INT), value :: istride
  integer(C_INT), value :: idist
  complex(C_FLOAT_COMPLEX), dimension(*), intent(out) :: out
  integer(C_INT), dimension(*), intent(in) :: onembed
  integer(C_INT), value :: ostride
  integer(C_INT), value :: odist
  integer(C_INT), value :: flags
end function fftwf_plan_many_dft_r2c
```

Returns

This function returns an `fftwf_plan` object that can be used in subsequent *fftwf_execute*`()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwf_execute_dft_r2c*`()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of the FFT problem to be solved

Constraint: rank >= 1

`n` Input parameter

`n` is `const int`

`n` is an array of dimension `[rank]`.

`n` contains the size of each dimension of the FFT to be solved

Constraint: `n[i]` >= 1 for i in 1, rank

`howmany` Input parameter

`howmany` is `int`

`howmany` is the number of multi-dimensional FFT transforms to be computed

Constraint: `howmany` >= 1

`in` Input parameter

`in` is `float`

`in` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will have been filled with the input sequence on which to perform the FFT. The input data for set `k` starts at address `in+k*idist`.

Storage: Data in arrays will be stored in row-major order.

`inembed` Input parameter

`inembed` is `const int`

`inembed` is an array of dimension `[rank]` or `NULL`. This array denotes the dimensionality of a potentially larger space of which the input array, `in`, dimensions given by `n`, is a sub array.

Constraint: Either `inembed[i] >= n[i]` for `i` in 0, `rank-1` OR if `inembed == NULL`, then `inembed` is assumed to have the same dimensions as `n`

`istride` Input parameter

`istride` is `int`

`istride` is the distance between successive elements of the input data

`idist` Input parameter

`idist` is `int`

`idist` is the distance between the start of each of the `howmany` transforms in the input data

`out` Input parameter

`out` is `fftwf_complex*`

`out` is a pointer to a previously allocated array of `fftwf_complex` data. At `fftwf_execute()` time this will be filled with the resulting transform coefficients The output data for set `k` starts at address `out+k*odist`.

Storage: Data in arrays will be stored in row-major order.

`onembed` Input parameter

`onembed` is `const int`

`onembed` is an array of dimension `[rank]` or `NULL`. This array denotes the dimensionality of a potentially larger space of which the input array, `out`, dimensions given by `n`, is a sub array.

Constraint: Either `onembed[i] >= n[i]` for `i` in 0, `rank-1` OR if `onembed == NULL`, then `onembed` is assumed to have the same dimensions as `n`

`ostride` Input parameter

`ostride` is `int`

`ostride` is the distance between successive elements of the output data

`odist` Input parameter

`odist` is `int`

`odist` is the distance between the start of each of the `howmany` transforms in the output data

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwf_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.4.19 fftwh_plan_dft_r2c

`fftwh_plan_dft_r2c` is a basic FFTW interface call for planning an *n*-dimensional real-to-complex FFT operation on a single, contiguous data sequence.

Note: `in` and `out` can be the same pointer for an "in-place" transform.

Syntax

C specification:

```
#include "fftw3.h"

fftwh_plan fftwh_plan_dft_r2c(int rank, const int *n, __fp16 *in,
                              fftwh_complex *out, unsigned flags);
```

Returns

This function returns an `fftwh_plan` object that can be used in subsequent *fftwh_execute*`()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwh_execute_dft_r2c*`()`.

Parameters

rank Input parameter

rank is int

The number of dimensions of the FFT problem to be solved

Constraint: rank >= 1

n Input parameter

n is a const int*.

n is an array of dimension [rank].

n contains the size of each dimension of the FFT to be solved

Constraint: n[i] >=1 for i in 1, rank

in Input parameter

in is __fp16

in is a pointer to a previously allocated array of __fp16 data. At fftwh_execute() time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

out Input parameter

out is fftwh_complex*

out is a pointer to a previously allocated array of fftwh_complex data. At fftwh_execute() time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

flags Input parameter

flags is unsigned

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in flags are:

- FFTW_ESTIMATE – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- FFTW_MEASURE – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- FFTW_PATIENT – Produce a plan, testing more options than FFTW_MEASURE. Input and output data might be overwritten.
- FFTW_EXHAUSTIVE – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- FFTW_WISDOM_ONLY – Produce a plan only if the requested case has existing wisdom, otherwise return NULL.
- FFTW_PRESERVE_INPUT – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- FFTW_DESTROY_INPUT – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- FFTW_CONSERVE_MEMORY – Request that extra memory is not created. This might be ignored.
- FFTW_UNALIGNED – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwh_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.4.20 fftwh_plan_dft_r2c_1d

`fftwh_plan_dft_r2c_1d` is a basic FFTW interface call for planning a 1-d real-to-complex FFT operation on a single, contiguous data sequence.

Note: `in` and `out` can be the same pointer for an "in-place" transform.

Syntax

C specification:

```
#include "fftw3.h"

fftwh_plan fftwh_plan_dft_r2c_1d(int n0, __fp16 *in, fftwh_complex *out,
                                 unsigned flags);
```

Returns

This function returns an `fftwh_plan` object that can be used in subsequent *fftwh_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwh_execute_dft_r2c* `()`.

Parameters

`n0` Input parameter

`n0` is `int`

This is the length of the FFT sequence

Constraint: `n0` >= 1

`in` Input parameter

`in` is `__fp16`

`in` is a pointer to a previously allocated array of `__fp16` data. At `fftwh_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `fftwh_complex*`

`out` is a pointer to a previously allocated array of `fftwh_complex` data. At `fftwh_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwh_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.4.21 fftwh_plan_dft_r2c_2d

`fftwh_plan_dft_r2c_2d` is a basic FFTW interface call for planning a 2-d real-to-complex FFT operation on a single, contiguous data sequence.

Note: `in` and `out` can be the same pointer for an "in-place" transform.

Syntax

C specification:

```
#include "fftw3.h"

fftwh_plan fftwh_plan_dft_r2c_2d(int n0, int n1, __fp16 *in,
                                 fftwh_complex *out, unsigned flags);
```

Returns

This function returns an `fftwh_plan` object that can be used in subsequent *fftwh_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwh_execute_dft_r2c* `()`.

Parameters

n0 Input parameter

`n0` is `int`

This is the length of the first dimension of the FFT sequence

Constraint: `n0` >= 1

n1 Input parameter

`n1` is `int`

This is the length of the second dimension of the FFT sequence

Constraint: `n1` >= 1

in Input parameter

`in` is `__fp16`

`in` is a pointer to a previously allocated array of `__fp16` data. At `fftwh_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

out Input parameter

`out` is fftwh_complex

`out` is a pointer to a previously allocated array of `fftwh_complex` data. At `fftwh_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

flags Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwh_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.4.22 fftwh_plan_dft_r2c_3d

`fftwh_plan_dft_r2c_3d` is a basic FFTW interface call for planning a 3-d real-to-complex FFT operation on a single, contiguous data sequence.

Note: `in` and `out` can be the same pointer for an "in-place" transform.

Syntax

C specification:

```
#include "fftw3.h"

fftwh_plan fftwh_plan_dft_r2c_3d(int n0, int n1, int n2, __fp16 *in,
                                 fftwh_complex *out, unsigned flags);
```

Returns

This function returns an `fftwh_plan` object that can be used in subsequent *fftwh_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwh_execute_dft_r2c* `()`.

Parameters

n0 Input parameter

`n0` is `int`

This is the length of the first dimension of the FFT sequence

Constraint: `n0` >= 1

n1 Input parameter

`n1` is `int`

This is the length of the second dimension of the FFT sequence

Constraint: `n1` >= 1

n2 Input parameter

`n2` is `int`

This is the length of the third dimension of the FFT sequence

Constraint: `n2` >= 1

in Input parameter

`in` is __fp16

`in` is a pointer to a previously allocated array of `__fp16` data. At `fftwh_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

out Input parameter

`out` is `fftwh_complex*`

`out` is a pointer to a previously allocated array of `fftwh_complex` data. At `fftwh_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

flags Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwh_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.4.23 fftwh_plan_guru_dft_r2c

`fftwh_plan_guru_dft_r2c` is a guru FFTW interface call for planning an *n*-dimensional real-to-complex FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner.

Syntax

C specification:

```
#include "fftw3.h"

fftwh_plan fftwh_plan_guru_dft_r2c(int rank, const fftwh_iodim *dims,
                                   int howmany_rank,
```

(continues on next page)

(continued from previous page)

```
                                const fftwh_iodim *howmany_dims,
                                __fp16 *in, fftwh_complex *out,
                                unsigned flags);
```

Returns

This function returns an `fftwh_plan` object that can be used in subsequent *fftwh_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwh_execute_dft_r2c* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

`dims` Input parameter

`dims` is `const fftwh_iodim *`

`dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the length of the `i` th dimension.
- `int is` – the stride between successive elements in the input vector in the `i` th dimension.
- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: `dims.n[i]` >= 1 for i in 0, `rank-1`

`howmany_rank` Input parameter

`howmany_rank` is `int`

`howmany_rank` is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then `howmany_rank` needs only be 1.

Constraint: `howmany` >= 0

Note: If `howmany_rank=0`, then a single FFT transform is computed.

`howmany_dims` Input parameter

`howmany_dims` is `const fftwh_iodim *`

`howmany_dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the number of transforms to be computed of the `i` th dimension.of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `int is` – the number of memory elements, of type `fftwh_complex`, between the start of transforms in the `i` th dimension. of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the `howmany_dims.n[]` entries. For example, if `howmany_rank=1`, then `howmany_dims[0].n` is the memory distance between successive transforms.

Constraint: `howmany_dims.n[i]` >= 1 for i in 0, `howmany_rank-1`

`in` Input parameter

`in` is `__fp16`

`in` is a pointer to a previously allocated array of `__fp16` data. At `fftwh_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `fftwh_complex*`

`out` is a pointer to a previously allocated array of `fftwh_complex` data. At `fftwh_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwh_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.4.24 fftwh_plan_guru_split_dft_r2c

`fftwh_plan_guru_split_dft_r2c` is a guru FFTW interface call for planning an *n*-dimensional real-to-complex FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner. Input and output data is stored in separate arrays for real and imaginary parts.

Syntax

C specification:

```
#include "fftw3.h"

fftwh_plan fftwh_plan_guru_split_dft_r2c(int rank, const fftwh_iodim *dims,
                                         int howmany_rank,
                                         const fftwh_iodim *howmany_dims,
                                         __fp16 *in, __fp16 *ro,
                                         __fp16 *io, unsigned flags);
```

Returns

This function returns an `fftwh_plan` object that can be used in subsequent *fftwh_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwh_execute_dft_r2c* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

`dims` Input parameter

`dims` is `const fftwh_iodim *`

`dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the length of the `i` th dimension.
- `int is` – the stride between successive elements in the input vector in the `i` th dimension.
- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: `dims.n[i]` >= 1 for i in 0, `rank-1`

`howmany_rank` Input parameter

`howmany_rank` is `int`

`howmany_rank` is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then `howmany_rank` needs only be 1.

Constraint: `howmany` >= 0

Note: If `howmany_rank=0`, then a single FFT transform is computed.

howmany_dims Input parameter

howmany_dims is const fftwh_iodim *

howmany_dims is an array of structures, dimension [rank], where the members of dims[i], for i in 0, rank-1, are defined to be:

- int n – the number of transforms to be computed of the i th dimension.of the howmany_rank-dimensional space mapped by the transforms to be computed.
- int is – the number of memory elements, of type fftwh_complex, between the start of transforms in the i th dimension. of the howmany_rank-dimensional space mapped by the transforms to be computed.
- int os – the stride between successive elements in the output vector in the i th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the howmany_dims.n[] entries. For example, if howmany_rank=1, then howmany_dims[0].n is the memory distance between successive transforms.

Constraint: howmany_dims.n[i] >= 1 for i in 0, howmany_rank-1

in Input parameter

in is __fp16 *

in is a pointer to a previously allocated array of __fp16 data. At fftwh_execute() time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

ro Input parameter

ro is __fp16 *

ro is a pointer to a previously allocated array of __fp16 data. At fftwh_execute() time this will be filled with the real parts of the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

io Input parameter

io is __fp16 *

io is a pointer to a previously allocated array of __fp16 data. At fftwh_execute() time this will be filled with the imaginary parts of the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

flags Input parameter

flags is unsigned

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in flags are:

- FFTW_ESTIMATE – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- FFTW_MEASURE – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- FFTW_PATIENT – Produce a plan, testing more options than FFTW_MEASURE. Input and output data might be overwritten.
- FFTW_EXHAUSTIVE – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.

- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwh_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.4.25 fftwh_plan_guru64_dft_r2c

`fftwh_plan_guru64_dft_r2c` is a guru FFTW interface call for planning an *n*-dimensional real-to-complex FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner.

Unlike *fftwh_plan_guru_dft_r2c*, `fftwh_plan_guru64_dft_r2c` allows selected parameters to be specified as 64-bit integers. For Arm Performance Libraries, this parameter size difference is true in both `armpl*_lp64` and `armpl_ilp64` variants. Therefore, the parameter size difference is independent of the integer size used in the rest of the library interface.

Syntax

C specification:

```
#include "fftw3.h"

fftwh_plan fftwh_plan_guru64_dft_r2c(int rank, const fftwh_iodim64 *dims,
                                     int howmany_rank,
                                     const fftwh_iodim64 *howmany_dims,
                                     __fp16 *in, fftwh_complex *out,
                                     unsigned flags);
```

Returns

This function returns an `fftwh_plan` object that can be used in subsequent *fftwh_execute*`()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwh_execute_dft_r2c*`()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

dims Input parameter

dims is const fftwh_iodim64 *

dims is an array of structures, dimension [rank], where the members of dims[i], for i in 0, rank-1, are defined to be:

- ptrdiff_t n – the length of the i th dimension.
- ptrdiff_t is – the stride between successive elements in the input vector in the i th dimension.
- ptrdiff_t os – the stride between successive elements in the output vector in the i th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: dims.n[i] >= 1 for i in 0, rank-1

howmany_rank Input parameter

howmany_rank is int

howmany_rank is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then howmany_rank needs only be 1.

Constraint: howmany >= 0

Note: If howmany_rank=0, then a single FFT transform is computed.

howmany_dims Input parameter

howmany_dims is const fftwh_iodim64 *

howmany_dims is an array of structures, dimension [rank], where the members of dims[i], for i in 0, rank-1, are defined to be:

- ptrdiff_t n – the number of transforms to be computed of the i th dimension.of the howmany_rank-dimensional space mapped by the transforms to be computed.
- ptrdiff_t is – the number of memory elements, of type fftwh_complex, between the start of transforms in the i th dimension. of the howmany_rank-dimensional space mapped by the transforms to be computed.
- ptrdiff_t os – the stride between successive elements in the output vector in the i th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the howmany_dims.n[] entries. For example, if howmany_rank=1, then howmany_dims[0].n is the memory distance between successive transforms.

Constraint: howmany_dims.n[i] >= 1 for i in 0, howmany_rank-1

in Input parameter

in is __fp16

in is a pointer to a previously allocated array of __fp16 data. At fftwh_execute() time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

out Input parameter

out is fftwh_complex*

out is a pointer to a previously allocated array of fftwh_complex data. At fftwh_execute() time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

flags Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwh_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.4.26 fftwh_plan_guru64_split_dft_r2c

`fftwh_plan_guru64_split_dft_r2c` is a guru FFTW interface call for planning an *n*-dimensional real-to-complex FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner. Input and output data is stored in separate arrays for real and imaginary parts.

Unlike *fftwh_plan_guru_split_dft_r2c*, `fftwh_plan_guru64_split_dft_r2c` allows selected parameters to be specified as 64-bit integers. For Arm Performance Libraries, this parameter size difference is true in both `armpl*_lp64` and `armpl_ilp64` variants. Therefore, the parameter size difference is independent of the integer size used in the rest of the library interface.

Syntax

C specification:

```
#include "fftw3.h"

fftwh_plan fftwh_plan_guru_split_dft_r2c(int rank, const fftwh_iodim64 *dims,
                                         int howmany_rank,
```

(continues on next page)

(continued from previous page)

```
                                        const fftwh_iodim64 *howmany_dims,
                                        __fp16 *in, __fp16 *ro,
                                        __fp16 *io, unsigned flags);
```

Returns

This function returns an `fftwh_plan` object that can be used in subsequent *fftwh_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwh_execute_dft_r2c* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

`dims` Input parameter

`dims` is `const fftwh_iodim64 *`

`dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `ptrdiff_t n` – the length of the `i` th dimension.
- `ptrdiff_t is` – the stride between successive elements in the input vector in the `i` th dimension.
- `ptrdiff_t os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: `dims.n[i]` >= 1 for i in 0, `rank-1`

`howmany_rank` Input parameter

`howmany_rank` is `int`

`howmany_rank` is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then `howmany_rank` needs only be 1.

Constraint: `howmany` >= 0

Note: If `howmany_rank=0`, then a single FFT transform is computed.

`howmany_dims` Input parameter

`howmany_dims` is `const fftwh_iodim64 *`

`howmany_dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `ptrdiff_t n` – the number of transforms to be computed of the `i` th dimension.of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `ptrdiff_t is` – the number of memory elements, of type `fftwh_complex`, between the start of transforms in the `i` th dimension. of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `ptrdiff_t os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the `howmany_dims.n[]` entries. For example, if `howmany_rank=1`, then `howmany_dims[0].n` is the memory distance between successive transforms.

Constraint: `howmany_dims.n[i]` >= 1 for i in 0, `howmany_rank-1`

`in` Input parameter

`in` is `__fp16 *`

`in` is a pointer to a previously allocated array of `__fp16` data. At `fftwh_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`ro` Input parameter

`ro` is `__fp16 *`

`ro` is a pointer to a previously allocated array of `__fp16` data. At `fftwh_execute()` time this will be filled with the real parts of the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`io` Input parameter

`io` is `__fp16 *`

`io` is a pointer to a previously allocated array of `__fp16` data. At `fftwh_execute()` time this will be filled with the imaginary parts of the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwh_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.4.27 fftwh_plan_many_dft_r2c

`fftwh_plan_many_dft_r2c` is an advanced FFTW interface call for planning an *n*-dimensional real-to-complex FFT operation on `howmany` sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner.

For more information on the use of advanced planning calls see *How to choose which FFT routine you need*.

Syntax

C specification:

```
#include "fftw3.h"

fftwh_plan fftwh_plan_many_dft_r2c(int rank, const int *n, int howmany,
                                   __fp16 *in, const int *inembed,
                                   int istride, int idist, fftwh_complex *out,
                                   const int *onembed, int ostride, int odist,
                                   unsigned flags);
```

Returns

This function returns an `fftwh_plan` object that can be used in subsequent *fftwh_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwh_execute_dft_r2c* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of the FFT problem to be solved

Constraint: rank >= 1

`n` Input parameter

`n` is `const int`

`n` is an array of dimension `[rank]`.

`n` contains the size of each dimension of the FFT to be solved

Constraint: `n[i]` >= 1 for i in 1, rank

`howmany` Input parameter

`howmany` is `int`

`howmany` is the number of multi-dimensional FFT transforms to be computed

Constraint: `howmany` >= 1

`in` Input parameter

`in` is `__fp16`

`in` is a pointer to a previously allocated array of `__fp16` data. At `fftwh_execute()` time this will have been filled with the input sequence on which to perform the FFT. The input data for set `k` starts at address `in+k*idist`.

Storage: Data in arrays will be stored in row-major order.

`inembed` Input parameter

`inembed` is `const int`

`inembed` is an array of dimension `[rank]` or `NULL`. This array denotes the dimensionality of a potentially larger space of which the input array, `in`, dimensions given by `n`, is a sub array.

Constraint: Either `inembed[i] >= n[i]` for `i` in 0, `rank-1` OR if `inembed == NULL`, then `inembed` is assumed to have the same dimensions as `n`

`istride` Input parameter

`istride` is `int`

`istride` is the distance between successive elements of the input data

`idist` Input parameter

`idist` is `int`

`idist` is the distance between the start of each of the `howmany` transforms in the input data

`out` Input parameter

`out` is `fftwh_complex*`

`out` is a pointer to a previously allocated array of `fftwh_complex` data. At `fftwh_execute()` time this will be filled with the resulting transform coefficients The output data for set `k` starts at address `out+k*odist`.

Storage: Data in arrays will be stored in row-major order.

`onembed` Input parameter

`onembed` is `const int`

`onembed` is an array of dimension `[rank]` or `NULL`. This array denotes the dimensionality of a potentially larger space of which the input array, `out`, dimensions given by `n`, is a sub array.

Constraint: Either `onembed[i] >= n[i]` for `i` in 0, `rank-1` OR if `onembed == NULL`, then `onembed` is assumed to have the same dimensions as `n`

`ostride` Input parameter

`ostride` is `int`

`ostride` is the distance between successive elements of the output data

`odist` Input parameter

`odist` is `int`

`odist` is the distance between the start of each of the `howmany` transforms in the output data

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.

- FFTW_PATIENT – Produce a plan, testing more options than FFTW_MEASURE. Input and output data might be overwritten.
- FFTW_EXHAUSTIVE – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- FFTW_WISDOM_ONLY – Produce a plan only if the requested case has existing wisdom, otherwise return NULL.
- FFTW_PRESERVE_INPUT – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- FFTW_DESTROY_INPUT – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- FFTW_CONSERVE_MEMORY – Request that extra memory is not created. This might be ignored.
- FFTW_UNALIGNED – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by FFTW_MEASURE, FFTW_PATIENT and FFTW_EXHAUSTIVE will be limited when using *fftwh_set_timelimit*.
- If multiple options for planning rigor are included in flags, then the highest level of planning requested will be used.

5.5 FFT complex to real functions

5.5.1 fftw_plan_dft_c2r

fftw_plan_dft_c2r is a basic FFTW interface call for planning an *n*-dimensional complex-to-real FFT operation on a single, contiguous data sequence.

Note: in and out can be the same pointer for an "in-place" transform.

Syntax

C specification:

```
#include "fftw3.h"

fftw_plan fftw_plan_dft_c2r(int rank, const int *n, fftw_complex *in,
                            double *out, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_plan_dft_c2r(rank,n,in,out,flags) &
    bind(C, name='fftw_plan_dft_c2r')
  integer(C_INT), value :: rank
  integer(C_INT), dimension(*), intent(in) :: n
  complex(C_DOUBLE_COMPLEX), dimension(*), intent(out) :: in
  real(C_DOUBLE), dimension(*), intent(out) :: out
  integer(C_INT), value :: flags
end function fftw_plan_dft_c2r
```

Returns

An `fftw_plan` object is returned that can be used in subsequent *fftw_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftw_execute_dft_c2r* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of the FFT problem to be solved

Constraint: `rank` >= 1

`n` Input parameter

`n` is `const int*`.

`n` is an array of dimension `[rank]`.

`n` contains the size of each dimension of the FFT to be solved

Constraint: `n[i]` >= 1 for i in 1, `rank`

`in` Input parameter

in is fftw_complex

`in` is a pointer to a previously allocated array of `fftw_complex` data. At `fftw_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `double*`

`out` is a pointer to a previously allocated array of `double` data. At `fftw_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.

- FFTW_DESTROY_INPUT – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- FFTW_CONSERVE_MEMORY – Request that extra memory is not created. This might be ignored.
- FFTW_UNALIGNED – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by FFTW_MEASURE, FFTW_PATIENT and FFTW_EXHAUSTIVE will be limited when using *fftw_set_timelimit*.
- If multiple options for planning rigor are included in flags, then the highest level of planning requested will be used.

5.5.2 fftw_plan_dft_c2r_1d

fftw_plan_dft_c2r_1d is a basic FFTW interface call for planning a 1-d complex-to-real FFT operation on a single, contiguous data sequence.

Note: in and out can be the same pointer for an "in-place" transform.

Syntax

C specification:

```
#include "fftw3.h"

fftw_plan fftw_plan_dft_c2r_1d(int n0, fftw_complex *in, double *out,
                               unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_plan_dft_c2r_1d(n,in,out,flags) &
    bind(C, name='fftw_plan_dft_c2r_1d')
  integer(C_INT), value :: n
  complex(C_DOUBLE_COMPLEX), dimension(*), intent(out) :: in
  real(C_DOUBLE), dimension(*), intent(out) :: out
  integer(C_INT), value :: flags
end function fftw_plan_dft_c2r_1d
```

Returns

An fftw_plan object is returned that can be used in subsequent *fftw_execute*() calls on the previously supplied in and out arrays, or using different arrays via *fftw_execute_dft_c2r*().

Parameters

n0 Input parameter

n0 is int

This is the length of the FFT sequence

Constraint: `n0` >= 1

`in` Input parameter

`in` is `fftw_complex*`

`in` is a pointer to a previously allocated array of `fftw_complex` data. At `fftw_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `double*`

`out` is a pointer to a previously allocated array of `double` data. At `fftw_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftw_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.5.3 fftw_plan_dft_c2r_2d

`fftw_plan_dft_c2r_2d` is a basic FFTW interface call for planning a 2-d complex-to-real FFT operation on a single, contiguous data sequence.

Note: `in` and `out` can be the same pointer for an "in-place" transform.

Syntax

C specification:

```
#include "fftw3.h"

fftw_plan fftw_plan_dft_c2r_2d(int n0, int n1, fftw_complex *in,
                               double *out, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_plan_dft_c2r_2d(n0,n1,in,out,flags) &
    bind(C, name='fftw_plan_dft_c2r_2d')
  integer(C_INT), value :: n0
  integer(C_INT), value :: n1
  complex(C_DOUBLE_COMPLEX), dimension(*), intent(out) :: in
  real(C_DOUBLE), dimension(*), intent(out) :: out
  integer(C_INT), value :: flags
end function fftw_plan_dft_c2r_2d
```

Returns

An `fftw_plan` object is returned that can be used in subsequent *fftw_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftw_execute_dft_c2r* `()`.

Parameters

`n0` Input parameter

`n0` is `int`

This is the length of the first dimension of the FFT sequence

Constraint: `n0` >= 1

`n1` Input parameter

`n1` is `int`

This is the length of the second dimension of the FFT sequence

Constraint: `n1` >= 1

`in` Input parameter

`in` is `fftw_complex*`

`in` is a pointer to a previously allocated array of `fftw_complex` data. At `fftw_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

out is double

`out` is a pointer to a previously allocated array of `double` data. At `fftw_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftw_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.5.4 fftw_plan_dft_c2r_3d

`fftw_plan_dft_c2r_3d` is a basic FFTW interface call for planning a 3-d complex-to-real FFT operation on a single, contiguous data sequence.

Note: `in` and `out` can be the same pointer for an "in-place" transform.

Syntax

C specification:

```
#include "fftw3.h"

fftw_plan fftw_plan_dft_c2r_3d(int n0, int n1, int n2, fftw_complex *in,
                               double *out, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_plan_dft_c2r_3d(n0,n1,n2,in,out,flags) &
    bind(C, name='fftw_plan_dft_c2r_3d')
  integer(C_INT), value :: n0
  integer(C_INT), value :: n1
  integer(C_INT), value :: n2
  complex(C_DOUBLE_COMPLEX), dimension(*), intent(out) :: in
  real(C_DOUBLE), dimension(*), intent(out) :: out
  integer(C_INT), value :: flags
end function fftw_plan_dft_c2r_3d
```

Returns

An `fftw_plan` object is returned that can be used in subsequent *fftw_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftw_execute_dft_c2r* `()`.

Parameters

`n0` Input parameter

`n0` is `int`

This is the length of the first dimension of the FFT sequence

Constraint: `n0` >= 1

`n1` Input parameter

`n1` is `int`

This is the length of the second dimension of the FFT sequence

Constraint: `n1` >= 1

`n2` Input parameter

`n2` is `int`

This is the length of the third dimension of the FFT sequence

Constraint: `n2` >= 1

`in` Input parameter

`in` is fftw_complex

`in` is a pointer to a previously allocated array of `fftw_complex` data. At `fftw_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `double*`

`out` is a pointer to a previously allocated array of `double` data. At `fftw_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftw_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.5.5 fftw_plan_guru_dft_c2r

`fftw_plan_guru_dft_c2r` is a guru FFTW interface call for planning an *n*-dimensional complex-to-real FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner.

Syntax

C specification:

```
#include "fftw3.h"

fftw_plan fftw_plan_guru_dft_c2r(int rank, const fftw_iodim *dims,
                                 int howmany_rank,
                                 const fftw_iodim *howmany_dims,
                                 fftw_complex *in, double *out,
                                 unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_plan_guru_dft_c2r(rank,dims,howmany_rank, &
    howmany_dims,in,out,flags) &
    bind(C, name='fftw_plan_guru_dft_c2r')
  integer(C_INT), value :: rank
  type(fftw_iodim), dimension(*), intent(in) :: dims
```

(continues on next page)

(continued from previous page)

```
  integer(C_INT), value :: howmany_rank
  type(fftw_iodim), dimension(*), intent(in) :: howmany_dims
  complex(C_DOUBLE_COMPLEX), dimension(*), intent(out) :: in
  real(C_DOUBLE), dimension(*), intent(out) :: out
  integer(C_INT), value :: flags
end function fftw_plan_guru_dft_c2r
```

Returns

An `fftw_plan` object is returned that can be used in subsequent *fftw_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftw_execute_dft* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

`dims` Input parameter

`dims` is `const fftw_iodim *`

`dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the length of the `i` th dimension.
- `int is` – the stride between successive elements in the input vector in the `i` th dimension.
- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: `dims.n[i]` >= 1 for i in 0, `rank-1`

`howmany_rank` Input parameter

`howmany_rank` is `int`

`howmany_rank` is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then `howmany_rank` needs only be 1.

Constraint: `howmany` >= 0

Note: If `howmany_rank=0`, then a single FFT transform is computed.

`howmany_dims` Input parameter

`howmany_dims` is `const fftw_iodim *`

`howmany_dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the number of transforms to be computed of the `i` th dimension.of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `int is` – the number of memory elements, of type `fftw_complex`, between the start of transforms in the `i` th dimension. of the `howmany_rank`-dimensional space mapped by the transforms to be computed.

- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the `howmany_dims.n[]` entries. For example, if `howmany_rank=1`, then `howmany_dims[0].n` is the memory distance between successive transforms.

Constraint: `howmany_dims.n[i]` >= 1 for i in 0, `howmany_rank-1`

`in` Input parameter

`in` is `fftw_complex*`

`in` is a pointer to a previously allocated array of `fftw_complex` data. At `fftw_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `fftw_complex*`

`out` is a pointer to a previously allocated array of `double` data. At `fftw_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftw_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.5.6 fftw_plan_guru_split_dft_c2r

`fftw_plan_guru_split_dft_c2r` is a guru FFTW interface call for planning an *n*-dimensional complex-to-real FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner. Input and output data is stored in separate arrays for real and imaginary parts.

Syntax

C specification:

```
#include "fftw3.h"

fftw_plan fftw_plan_guru_split_dft_c2r(int rank, const fftw_iodim *dims,
                                       int howmany_rank,
                                       const fftw_iodim *howmany_dims,
                                       double *ri, double *ii, double *out,
                                       unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_plan_guru_split_dft_c2r(rank,dims, &
    howmany_rank,howmany_dims,ri,ii,out,flags) &
    bind(C, name='fftw_plan_guru_split_dft_c2r')
  integer(C_INT), value :: rank
  type(fftw_iodim), dimension(*), intent(in) :: dims
  integer(C_INT), value :: howmany_rank
  type(fftw_iodim), dimension(*), intent(in) :: howmany_dims
  real(C_DOUBLE), dimension(*), intent(out) :: ri
  real(C_DOUBLE), dimension(*), intent(out) :: ii
  real(C_DOUBLE), dimension(*), intent(out) :: out
  integer(C_INT), value :: flags
end function fftw_plan_guru_split_dft_c2r
```

Returns

An `fftw_plan` object is returned that can be used in subsequent *fftw_execute*`()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftw_execute_dft*`()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

`dims` Input parameter

`dims` is `const fftw_iodim *`

`dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the length of the `i` th dimension.
- `int is` – the stride between successive elements in the input vector in the `i` th dimension.

- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: `dims.n[i]` >= 1 for i in 0, `rank-1`

howmany_rank Input parameter

`howmany_rank` is `int`

`howmany_rank` is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then `howmany_rank` needs only be 1.

Constraint: `howmany` >= 0

Note: If `howmany_rank=0`, then a single FFT transform is computed.

howmany_dims Input parameter

`howmany_dims` is `const fftw_iodim *`

`howmany_dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the number of transforms to be computed of the `i` th dimension.of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `int is` – the number of memory elements, of type `fftw_complex`, between the start of transforms in the `i` th dimension. of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the `howmany_dims.n[]` entries. For example, if `howmany_rank=1`, then `howmany_dims[0].n` is the memory distance between successive transforms.

Constraint: `howmany_dims.n[i]` >= 1 for i in 0, `howmany_rank-1`

ri Input parameter

`ri` is `double *`

`ri` is a pointer to a previously allocated array of `double` data. At `fftw_execute()` time this will have been filled with the real parts of the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

ii Input parameter

`ii` is `double *`

`ii` is a pointer to a previously allocated array of `double` data. At `fftw_execute()` time this will have been filled with the imaginary parts of the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

out Input parameter

`out` is `double *`

`out` is a pointer to a previously allocated array of `double` data. At `fftw_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

flags Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftw_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.5.7 fftw_plan_guru64_dft_c2r

`fftw_plan_guru64_dft_c2r` is a guru FFTW interface call for planning an *n*-dimensional complex-to-real FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner.

Unlike *fftw_plan_guru_dft_c2r*, `fftw_plan_guru64_dft_c2r` allows selected parameters to be specified as 64-bit integers. For Arm Performance Libraries, this parameter size difference is true in both `armpl*_lp64` and `armpl_ilp64` variants. Therefore, the parameter size difference is independent of the integer size used in the rest of the library interface.

Syntax

C specification:

```
#include "fftw3.h"

fftw_plan fftw_plan_guru64_dft_c2r(int rank, const fftw_iodim64 *dims,
                                   int howmany_rank,
                                   const fftw_iodim64 *howmany_dims,
                                   fftw_complex *in, double *out,
                                   unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_plan_guru64_dft_c2r(rank,dims,howmany_rank, &
    howmany_dims,in,out,flags) &
    bind(C, name='fftw_plan_guru64_dft_c2r')
  integer(C_INT), value :: rank
  type(fftw_iodim64), dimension(*), intent(in) :: dims
  integer(C_INT), value :: howmany_rank
  type(fftw_iodim64), dimension(*), intent(in) :: howmany_dims
  complex(C_DOUBLE_COMPLEX), dimension(*), intent(out) :: in
  real(C_DOUBLE), dimension(*), intent(out) :: out
  integer(C_INT), value :: flags
end function fftw_plan_guru64_dft_c2r
```

Returns

An `fftw_plan` object is returned that can be used in subsequent *fftw_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftw_execute_dft* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

`dims` Input parameter

`dims` is `const fftw_iodim64 *`

`dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `ptrdiff_t n` – the length of the `i` th dimension.
- `ptrdiff_t is` – the stride between successive elements in the input vector in the `i` th dimension.
- `ptrdiff_t os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: `dims.n[i]` >= 1 for i in 0, `rank-1`

`howmany_rank` Input parameter

`howmany_rank` is `int`

`howmany_rank` is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then `howmany_rank` needs only be 1.

Constraint: `howmany` >= 0

Note: If `howmany_rank=0`, then a single FFT transform is computed.

`howmany_dims` Input parameter

`howmany_dims` is `const fftw_iodim64 *`

`howmany_dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `ptrdiff_t n` – the number of transforms to be computed of the `i` th dimension.of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `ptrdiff_t is` – the number of memory elements, of type `fftw_complex`, between the start of transforms in the `i` th dimension. of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `ptrdiff_t os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the `howmany_dims.n[]` entries. For example, if `howmany_rank=1`, then `howmany_dims[0].n` is the memory distance between successive transforms.

Constraint: `howmany_dims.n[i]` >= 1 for i in 0, `howmany_rank-1`

`in` Input parameter

`in` is `fftw_complex*`

`in` is a pointer to a previously allocated array of `fftw_complex` data. At `fftw_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `double*`

`out` is a pointer to a previously allocated array of `double` data. At `fftw_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftw_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.5.8 fftw_plan_guru64_split_dft_c2r

`fftw_plan_guru64_split_dft_c2r` is a guru FFTW interface call for planning an *n*-dimensional complex-to-real FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner. Input and output data is stored in separate arrays for real and imaginary parts.

Unlike *fftw_plan_guru_split_dft_c2r*, `fftw_plan_guru64_split_dft_c2r` allows selected parameters to be specified as 64-bit integers. For Arm Performance Libraries, this parameter size difference is true in both `armpl*_lp64` and `armpl_ilp64` variants. Therefore, the parameter size difference is independent of the integer size used in the rest of the library interface.

Syntax

C specification:

```
#include "fftw3.h"

fftw_plan fftw_plan_guru_split_dft_c2r(int rank, const fftw_iodim64 *dims,
                                       int howmany_rank,
                                       const fftw_iodim64 *howmany_dims,
                                       double *ri, double *ii, double *out,
                                       unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_plan_guru64_split_dft_c2r(rank,dims, &
    howmany_rank,howmany_dims,ri,ii,out,flags) &
    bind(C, name='fftw_plan_guru64_split_dft_c2r')
  integer(C_INT), value :: rank
  type(fftw_iodim64), dimension(*), intent(in) :: dims
  integer(C_INT), value :: howmany_rank
  type(fftw_iodim64), dimension(*), intent(in) :: howmany_dims
  real(C_DOUBLE), dimension(*), intent(out) :: ri
  real(C_DOUBLE), dimension(*), intent(out) :: ii
  real(C_DOUBLE), dimension(*), intent(out) :: out
  integer(C_INT), value :: flags
end function fftw_plan_guru64_split_dft_c2r
```

Returns

An `fftw_plan` object is returned that can be used in subsequent *fftw_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftw_execute_dft* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

`dims` Input parameter

`dims` is `const fftw_iodim64 *`

`dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `ptrdiff_t n` – the length of the `i` th dimension.
- `ptrdiff_t is` – the stride between successive elements in the input vector in the `i` th dimension.
- `ptrdiff_t os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: `dims.n[i]` >= 1 for i in 0, `rank-1`

`howmany_rank` Input parameter

`howmany_rank` is `int`

`howmany_rank` is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then `howmany_rank` needs only be 1.

Constraint: `howmany` >= 0

Note: If `howmany_rank=0`, then a single FFT transform is computed.

`howmany_dims` Input parameter

`howmany_dims` is `const fftw_iodim64 *`

`howmany_dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `ptrdiff_t n` – the number of transforms to be computed of the `i` th dimension.of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `ptrdiff_t is` – the number of memory elements, of type `fftw_complex`, between the start of transforms in the `i` th dimension. of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `ptrdiff_t os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the `howmany_dims.n[]` entries. For example, if `howmany_rank=1`, then `howmany_dims[0].n` is the memory distance between successive transforms.

Constraint: `howmany_dims.n[i]` >= 1 for i in 0, `howmany_rank-1`

`ri` Input parameter

`ri` is `double *`

`ri` is a pointer to a previously allocated array of `double` data. At `fftw_execute()` time this will have been filled with the real parts of the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`ii` Input parameter

`ii` is `double *`

`ii` is a pointer to a previously allocated array of `double` data. At `fftw_execute()` time this will have been filled with the imaginary parts of the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

out Input parameter

`out` is `double *`

`out` is a pointer to a previously allocated array of `double` data. At `fftw_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

flags Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftw_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.5.9 fftw_plan_many_dft_c2r

`fftw_plan_many_dft_c2r` is an advanced FFTW interface call for planning an *n*-dimensional complex-to-real FFT operation on `howmany` sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner.

For more information on the use of advanced planning calls see *How to choose which FFT routine you need*.

Syntax

C specification:

```
#include "fftw3.h"

fftw_plan fftw_plan_many_dft_c2r(int rank, const int *n, int howmany,
                                 fftw_complex *in, const int *inembed,
                                 int istride, int idist, double *out,
                                 const int *onembed, int ostride,
                                 int odist, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_plan_many_dft_c2r(rank,n,howmany,in,inembed, &
    istride,idist,out,onembed,ostride,odist,flags) &
    bind(C, name='fftw_plan_many_dft_c2r')
  integer(C_INT), value :: rank
  integer(C_INT), dimension(*), intent(in) :: n
  integer(C_INT), value :: howmany
  complex(C_DOUBLE_COMPLEX), dimension(*), intent(out) :: in
  integer(C_INT), dimension(*), intent(in) :: inembed
  integer(C_INT), value :: istride
  integer(C_INT), value :: idist
  real(C_DOUBLE), dimension(*), intent(out) :: out
  integer(C_INT), dimension(*), intent(in) :: onembed
  integer(C_INT), value :: ostride
  integer(C_INT), value :: odist
  integer(C_INT), value :: flags
end function fftw_plan_many_dft_c2r
```

Returns

An `fftw_plan` object is returned that can be used in subsequent *fftw_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftw_execute_dft_c2r* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of the FFT problem to be solved

Constraint: rank >= 1

`n` Input parameter

`n` is `const int`

`n` is an array of dimension `[rank]`.

`n` contains the size of each dimension of the FFT to be solved

Constraint: `n[i]` >= 1 for i in 1, rank

`howmany` Input parameter

`howmany` is `int`

`howmany` is the number of multi-dimensional FFT transforms to be computed

Constraint: `howmany` >= 1

in Input parameter

in is fftw_complex*

in is a pointer to a previously allocated array of fftw_complex data. At fftw_execute() time this will have been filled with the input sequence on which to perform the FFT. The input data for set k starts at address in+k*idist.

Storage: Data in arrays will be stored in row-major order.

inembed Input parameter

inembed is const int

inembed is an array of dimension [rank] or NULL. This array denotes the dimensionality of a potentially larger space of which the input array, in, dimensions given by n, is a sub array.

Constraint: Either inembed[i] >= n[i] for i in 0, rank-1 OR if inembed == NULL, then inembed is assumed to have the same dimensions as n

istride Input parameter

istride is int

istride is the distance between successive elements of the input data

idist Input parameter

idist is int

idist is the distance between the start of each of the howmany transforms in the input data

out Input parameter

out is double*

out is a pointer to a previously allocated array of double data. At fftw_execute() time this will be filled with the resulting transform coefficients The output data for set k starts at address out+k*odist.

Storage: Data in arrays will be stored in row-major order.

onembed Input parameter

onembed is const int

onembed is an array of dimension [rank] or NULL. This array denotes the dimensionality of a potentially larger space of which the input array, out, dimensions given by n, is a sub array.

Constraint: Either onembed[i] >= n[i] for i in 0, rank-1 OR if onembed == NULL, then onembed is assumed to have the same dimensions as n

ostride Input parameter

ostride is int

ostride is the distance between successive elements of the output data

odist Input parameter

odist is int

odist is the distance between the start of each of the howmany transforms in the output data

flags Input parameter

flags is unsigned

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in flags are:

- FFTW_ESTIMATE – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.

- FFTW_MEASURE – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- FFTW_PATIENT – Produce a plan, testing more options than FFTW_MEASURE. Input and output data might be overwritten.
- FFTW_EXHAUSTIVE – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- FFTW_WISDOM_ONLY – Produce a plan only if the requested case has existing wisdom, otherwise return NULL.
- FFTW_PRESERVE_INPUT – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- FFTW_DESTROY_INPUT – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- FFTW_CONSERVE_MEMORY – Request that extra memory is not created. This might be ignored.
- FFTW_UNALIGNED – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by FFTW_MEASURE, FFTW_PATIENT and FFTW_EXHAUSTIVE will be limited when using *fftw_set_timelimit*.
- If multiple options for planning rigor are included in flags, then the highest level of planning requested will be used.

5.5.10 fftwf_plan_dft_c2r

fftwf_plan_dft_c2r is a basic FFTW interface call for planning an *n*-dimensional complex-to-real FFT operation on a single, contiguous data sequence.

Note: in and out can be the same pointer for an "in-place" transform.

Syntax

C specification:

```
#include "fftw3.h"

fftwf_plan fftwf_plan_dft_c2r(int rank, const int *n, fftwf_complex *in,
                              float *out, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_plan_dft_c2r(rank,n,in,out,flags) &
    bind(C, name='fftwf_plan_dft_c2r')
  integer(C_INT), value :: rank
  integer(C_INT), dimension(*), intent(in) :: n
  complex(C_FLOAT_COMPLEX), dimension(*), intent(out) :: in
  real(C_FLOAT), dimension(*), intent(out) :: out
  integer(C_INT), value :: flags
end function fftwf_plan_dft_c2r
```

Returns

This function returns an `fftwf_plan` object that can be used in subsequent *fftwf_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwf_execute_dft_c2r* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of the FFT problem to be solved

Constraint: `rank` >= 1

`n` Input parameter

`n` is `const int*`.

`n` is an array of dimension `[rank]`.

`n` contains the size of each dimension of the FFT to be solved

Constraint: `n[i]` >= 1 for i in 1, `rank`

`in` Input parameter

in is fftwf_complex

`in` is a pointer to a previously allocated array of `fftwf_complex` data. At `fftwf_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `float`

`out` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.

- FFTW_DESTROY_INPUT – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- FFTW_CONSERVE_MEMORY – Request that extra memory is not created. This might be ignored.
- FFTW_UNALIGNED – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by FFTW_MEASURE, FFTW_PATIENT and FFTW_EXHAUSTIVE will be limited when using *fftwf_set_timelimit*.
- If multiple options for planning rigor are included in flags, then the highest level of planning requested will be used.

5.5.11 fftwf_plan_dft_c2r_1d

fftwf_plan_dft_c2r_1d is a basic FFTW interface call for planning a 1-d complex-to-real FFT operation on a single, contiguous data sequence.

Note: in and out can be the same pointer for an "in-place" transform.

Syntax

C specification:

```
#include "fftw3.h"

fftwf_plan fftwf_plan_dft_c2r_1d(int n0, fftwf_complex *in, float *out,
                                 unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_plan_dft_c2r_1d(n,in,out,flags) &
    bind(C, name='fftwf_plan_dft_c2r_1d')
  integer(C_INT), value :: n
  complex(C_FLOAT_COMPLEX), dimension(*), intent(out) :: in
  real(C_FLOAT), dimension(*), intent(out) :: out
  integer(C_INT), value :: flags
end function fftwf_plan_dft_c2r_1d
```

Returns

This function returns an fftwf_plan object that can be used in subsequent *fftwf_execute*() calls on the previously supplied in and out arrays, or using different arrays via *fftwf_execute_dft_c2r*().

Parameters

n0 Input parameter

n0 is int

This is the length of the FFT sequence

Constraint: `n0` >= 1

`in` Input parameter

`in` is `fftwf_complex*`

`in` is a pointer to a previously allocated array of `fftwf_complex` data. At `fftwf_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `float`

`out` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwf_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.5.12 fftwf_plan_dft_c2r_2d

`fftwf_plan_dft_c2r_2d` is a basic FFTW interface call for planning a 2-d complex-to-real FFT operation on a single, contiguous data sequence.

Note: `in` and `out` can be the same pointer for an "in-place" transform.

Syntax

C specification:

```
#include "fftw3.h"

fftwf_plan fftwf_plan_dft_c2r_2d(int n0, int n1, fftwf_complex *in,
                                 float *out, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_plan_dft_c2r_2d(n0,n1,in,out,flags) &
    bind(C, name='fftwf_plan_dft_c2r_2d')
  integer(C_INT), value :: n0
  integer(C_INT), value :: n1
  complex(C_FLOAT_COMPLEX), dimension(*), intent(out) :: in
  real(C_FLOAT), dimension(*), intent(out) :: out
  integer(C_INT), value :: flags
end function fftwf_plan_dft_c2r_2d
```

Returns

This function returns an `fftwf_plan` object that can be used in subsequent *fftwf_execute*`()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwf_execute_dft_c2r*`()`.

Parameters

`n0` Input parameter

`n0` is `int`

This is the length of the first dimension of the FFT sequence

Constraint: `n0` >= 1

`n1` Input parameter

`n1` is `int`

This is the length of the second dimension of the FFT sequence

Constraint: `n1` >= 1

`in` Input parameter

`in` is `fftwf_complex*`

`in` is a pointer to a previously allocated array of `fftwf_complex` data. At `fftwf_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

out is float

`out` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwf_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.5.13 fftwf_plan_dft_c2r_3d

`fftwf_plan_dft_c2r_3d` is a basic FFTW interface call for planning a 3-d complex-to-real FFT operation on a single, contiguous data sequence.

Note: `in` and `out` can be the same pointer for an "in-place" transform.

Syntax

C specification:

```
#include "fftw3.h"

fftwf_plan fftwf_plan_dft_c2r_3d(int n0, int n1, int n2, fftwf_complex *in,
                                 float *out, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_plan_dft_c2r_3d(n0,n1,n2,in,out,flags) &
    bind(C, name='fftwf_plan_dft_c2r_3d')
  integer(C_INT), value :: n0
  integer(C_INT), value :: n1
  integer(C_INT), value :: n2
  complex(C_FLOAT_COMPLEX), dimension(*), intent(out) :: in
  real(C_FLOAT), dimension(*), intent(out) :: out
  integer(C_INT), value :: flags
end function fftwf_plan_dft_c2r_3d
```

Returns

This function returns an `fftwf_plan` object that can be used in subsequent *fftwf_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwf_execute_dft_c2r* `()`.

Parameters

`n0` Input parameter

`n0` is `int`

This is the length of the first dimension of the FFT sequence

Constraint: `n0` >= 1

`n1` Input parameter

`n1` is `int`

This is the length of the second dimension of the FFT sequence

Constraint: `n1` >= 1

`n2` Input parameter

`n2` is `int`

This is the length of the third dimension of the FFT sequence

Constraint: `n2` >= 1

`in` Input parameter

`in` is fftwf_complex

`in` is a pointer to a previously allocated array of `fftwf_complex` data. At `fftwf_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `float`

`out` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwf_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.5.14 fftwf_plan_guru_dft_c2r

`fftwf_plan_guru_dft_c2r` is a guru FFTW interface call for planning an *n*-dimensional complex-to-real FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner.

Syntax

C specification:

```
#include "fftw3.h"

fftwf_plan fftwf_plan_guru_dft_c2r(int rank, const fftwf_iodim *dims,
                                   int howmany_rank,
                                   const fftwf_iodim *howmany_dims,
                                   fftwf_complex *in, float *out,
                                   unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_plan_guru_dft_c2r(rank,dims,howmany_rank, &
    howmany_dims,in,out,flags) &
    bind(C, name='fftwf_plan_guru_dft_c2r')
  integer(C_INT), value :: rank
  type(fftwf_iodim), dimension(*), intent(in) :: dims
```

(continues on next page)

(continued from previous page)

```
  integer(C_INT), value :: howmany_rank
  type(fftwf_iodim), dimension(*), intent(in) :: howmany_dims
  complex(C_FLOAT_COMPLEX), dimension(*), intent(out) :: in
  real(C_FLOAT), dimension(*), intent(out) :: out
  integer(C_INT), value :: flags
end function fftwf_plan_guru_dft_c2r
```

Returns

This function returns an `fftwf_plan` object that can be used in subsequent *fftwf_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwf_execute_dft* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

`dims` Input parameter

`dims` is `const fftwf_iodim *`

`dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the length of the `i` th dimension.
- `int is` – the stride between successive elements in the input vector in the `i` th dimension.
- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: `dims.n[i]` >= 1 for i in 0, `rank-1`

`howmany_rank` Input parameter

`howmany_rank` is `int`

`howmany_rank` is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then `howmany_rank` needs only be 1.

Constraint: `howmany` >= 0

Note: If `howmany_rank=0`, then a single FFT transform is computed.

`howmany_dims` Input parameter

`howmany_dims` is `const fftwf_iodim *`

`howmany_dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the number of transforms to be computed of the `i` th dimension.of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `int is` – the number of memory elements, of type `fftwf_complex`, between the start of transforms in the `i` th dimension. of the `howmany_rank`-dimensional space mapped by the transforms to be computed.

- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the `howmany_dims.n[]` entries. For example, if `howmany_rank=1`, then `howmany_dims[0].n` is the memory distance between successive transforms.

Constraint: `howmany_dims.n[i]` >= 1 for i in 0, `howmany_rank-1`

`in` Input parameter

`in` is `fftwf_complex*`

`in` is a pointer to a previously allocated array of `fftwf_complex` data. At `fftwf_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `fftwf_complex*`

`out` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwf_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.5.15 fftwf_plan_guru_split_dft_c2r

`fftwf_plan_guru_split_dft_c2r` is a guru FFTW interface call for planning an *n*-dimensional complex-to-real FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner. Input and output data is stored in separate arrays for real and imaginary parts.

Syntax

C specification:

```
#include "fftw3.h"

fftwf_plan fftwf_plan_guru_split_dft_c2r(int rank, const fftwf_iodim *dims,
                                         int howmany_rank,
                                         const fftwf_iodim *howmany_dims,
                                         float *ri, float *ii, float *out,
                                         unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_plan_guru_split_dft_c2r(rank,dims, &
    howmany_rank,howmany_dims,ri,ii,out,flags) &
    bind(C, name='fftwf_plan_guru_split_dft_c2r')
  integer(C_INT), value :: rank
  type(fftwf_iodim), dimension(*), intent(in) :: dims
  integer(C_INT), value :: howmany_rank
  type(fftwf_iodim), dimension(*), intent(in) :: howmany_dims
  real(C_FLOAT), dimension(*), intent(out) :: ri
  real(C_FLOAT), dimension(*), intent(out) :: ii
  real(C_FLOAT), dimension(*), intent(out) :: out
  integer(C_INT), value :: flags
end function fftwf_plan_guru_split_dft_c2r
```

Returns

This function returns an `fftwf_plan` object that can be used in subsequent *fftwf_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwf_execute_dft* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

`dims` Input parameter

`dims` is `const fftwf_iodim *`

`dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the length of the `i` th dimension.
- `int is` – the stride between successive elements in the input vector in the `i` th dimension.

- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: `dims.n[i]` >= 1 for i in 0, `rank-1`

howmany_rank Input parameter

`howmany_rank` is `int`

`howmany_rank` is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then `howmany_rank` needs only be 1.

Constraint: `howmany` >= 0

Note: If `howmany_rank=0`, then a single FFT transform is computed.

howmany_dims Input parameter

`howmany_dims` is `const fftwf_iodim *`

`howmany_dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the number of transforms to be computed of the `i` th dimension.of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `int is` – the number of memory elements, of type `fftwf_complex`, between the start of transforms in the `i` th dimension. of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the `howmany_dims.n[]` entries. For example, if `howmany_rank=1`, then `howmany_dims[0].n` is the memory distance between successive transforms.

Constraint: `howmany_dims.n[i]` >= 1 for i in 0, `howmany_rank-1`

ri Input parameter

`ri` is `float *`

`ri` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will have been filled with the real parts of the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

ii Input parameter

`ii` is `float *`

`ii` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will have been filled with the imaginary parts of the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

out Input parameter

`out` is `float *`

`out` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

flags Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwf_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.5.16 fftwf_plan_guru64_dft_c2r

`fftwf_plan_guru64_dft_c2r` is a guru FFTW interface call for planning an *n*-dimensional complex-to-real FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner.

Unlike *fftwf_plan_guru_dft_c2r*, `fftwf_plan_guru64_dft_c2r` allows selected parameters to be specified as 64-bit integers. For Arm Performance Libraries, this parameter size difference is true in both `armpl*_lp64` and `armpl_ilp64` variants. Therefore, the parameter size difference is independent of the integer size used in the rest of the library interface.

Syntax

C specification:

```
#include "fftw3.h"

fftwf_plan fftwf_plan_guru64_dft_c2r(int rank, const fftwf_iodim64 *dims,
                                     int howmany_rank,
                                     const fftwf_iodim64 *howmany_dims,
                                     fftwf_complex *in, float *out,
                                     unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_plan_guru64_dft_c2r(rank,dims,howmany_rank, &
    howmany_dims,in,out,flags) &
    bind(C, name='fftwf_plan_guru64_dft_c2r')
  integer(C_INT), value :: rank
  type(fftwf_iodim64), dimension(*), intent(in) :: dims
  integer(C_INT), value :: howmany_rank
  type(fftwf_iodim64), dimension(*), intent(in) :: howmany_dims
  complex(C_FLOAT_COMPLEX), dimension(*), intent(out) :: in
  real(C_FLOAT), dimension(*), intent(out) :: out
  integer(C_INT), value :: flags
end function fftwf_plan_guru64_dft_c2r
```

Returns

This function returns an `fftwf_plan` object that can be used in subsequent *fftwf_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwf_execute_dft* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

`dims` Input parameter

`dims` is `const fftwf_iodim64 *`

`dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `ptrdiff_t n` – the length of the `i` th dimension.
- `ptrdiff_t is` – the stride between successive elements in the input vector in the `i` th dimension.
- `ptrdiff_t os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: `dims.n[i]` >= 1 for i in 0, `rank-1`

`howmany_rank` Input parameter

`howmany_rank` is `int`

`howmany_rank` is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then `howmany_rank` needs only be 1.

Constraint: `howmany` >= 0

Note: If `howmany_rank=0`, then a single FFT transform is computed.

`howmany_dims` Input parameter

`howmany_dims` is `const fftwf_iodim64 *`

`howmany_dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `ptrdiff_t n` – the number of transforms to be computed of the `i` th dimension.of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `ptrdiff_t is` – the number of memory elements, of type `fftwf_complex`, between the start of transforms in the `i` th dimension. of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `ptrdiff_t os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the `howmany_dims.n[]` entries. For example, if `howmany_rank=1`, then `howmany_dims[0].n` is the memory distance between successive transforms.

Constraint: `howmany_dims.n[i]` >= 1 for i in 0, `howmany_rank-1`

`in` Input parameter

`in` is `fftwf_complex*`

`in` is a pointer to a previously allocated array of `fftwf_complex` data. At `fftwf_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `float`

`out` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwf_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.5.17 fftwf_plan_guru64_split_dft_c2r

`fftwf_plan_guru64_split_dft_c2r` is a guru FFTW interface call for planning an *n*-dimensional complex-to-real FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner. Input and output data is stored in separate arrays for real and imaginary parts.

Unlike *fftwf_plan_guru_split_dft_c2r*, `fftwf_plan_guru64_split_dft_c2r` allows selected parameters to be specified as 64-bit integers. For Arm Performance Libraries, this parameter size difference is true in both `armpl*_lp64` and `armpl_ilp64` variants. Therefore, the parameter size difference is independent of the integer size used in the rest of the library interface.

Syntax

C specification:

```
#include "fftw3.h"

fftwf_plan fftwf_plan_guru_split_dft_c2r(int rank, const fftwf_iodim64 *dims,
                                         int howmany_rank,
                                         const fftwf_iodim64 *howmany_dims,
                                         float *ri, float *ii, float *out,
                                         unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_plan_guru64_split_dft_c2r(rank,dims, &
    howmany_rank,howmany_dims,ri,ii,out,flags) &
    bind(C, name='fftwf_plan_guru64_split_dft_c2r')
  integer(C_INT), value :: rank
  type(fftwf_iodim64), dimension(*), intent(in) :: dims
  integer(C_INT), value :: howmany_rank
  type(fftwf_iodim64), dimension(*), intent(in) :: howmany_dims
  real(C_FLOAT), dimension(*), intent(out) :: ri
  real(C_FLOAT), dimension(*), intent(out) :: ii
  real(C_FLOAT), dimension(*), intent(out) :: out
  integer(C_INT), value :: flags
end function fftwf_plan_guru64_split_dft_c2r
```

Returns

This function returns an `fftwf_plan` object that can be used in subsequent *fftwf_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwf_execute_dft* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

`dims` Input parameter

`dims` is `const fftwf_iodim64 *`

`dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `ptrdiff_t n` – the length of the `i` th dimension.
- `ptrdiff_t is` – the stride between successive elements in the input vector in the `i` th dimension.
- `ptrdiff_t os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: `dims.n[i]` >= 1 for i in 0, `rank-1`

`howmany_rank` Input parameter

`howmany_rank` is `int`

`howmany_rank` is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then `howmany_rank` needs only be 1.

Constraint: `howmany` >= 0

Note: If `howmany_rank=0`, then a single FFT transform is computed.

`howmany_dims` Input parameter

`howmany_dims` is `const fftwf_iodim64 *`

`howmany_dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `ptrdiff_t n` – the number of transforms to be computed of the `i` th dimension.of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `ptrdiff_t is` – the number of memory elements, of type `fftwf_complex`, between the start of transforms in the `i` th dimension. of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `ptrdiff_t os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the `howmany_dims.n[]` entries. For example, if `howmany_rank=1`, then `howmany_dims[0].n` is the memory distance between successive transforms.

Constraint: `howmany_dims.n[i]` >= 1 for i in 0, `howmany_rank-1`

`ri` Input parameter

`ri` is `float *`

`ri` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will have been filled with the real parts of the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`ii` Input parameter

`ii` is `float *`

`ii` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will have been filled with the imaginary parts of the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

out Input parameter

`out` is `float *`

`out` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

flags Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwf_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.5.18 fftwf_plan_many_dft_c2r

`fftwf_plan_many_dft_c2r` is an advanced FFTW interface call for planning an *n*-dimensional complex-to-real FFT operation on `howmany` sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner.

For more information on the use of advanced planning calls see *How to choose which FFT routine you need*.

Syntax

C specification:

```
#include "fftw3.h"

fftwf_plan fftwf_plan_many_dft_c2r(int rank, const int *n, int howmany,
                                   fftwf_complex *in, const int *inembed,
                                   int istride, int idist, float *out,
                                   const int *onembed, int ostride,
                                   int odist, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_plan_many_dft_c2r(rank,n,howmany,in,inembed, &
    istride,idist,out,onembed,ostride,odist,flags) &
    bind(C, name='fftwf_plan_many_dft_c2r')
  integer(C_INT), value :: rank
  integer(C_INT), dimension(*), intent(in) :: n
  integer(C_INT), value :: howmany
  complex(C_FLOAT_COMPLEX), dimension(*), intent(out) :: in
  integer(C_INT), dimension(*), intent(in) :: inembed
  integer(C_INT), value :: istride
  integer(C_INT), value :: idist
  real(C_FLOAT), dimension(*), intent(out) :: out
  integer(C_INT), dimension(*), intent(in) :: onembed
  integer(C_INT), value :: ostride
  integer(C_INT), value :: odist
  integer(C_INT), value :: flags
end function fftwf_plan_many_dft_c2r
```

Returns

This function returns an `fftwf_plan` object that can be used in subsequent *fftwf_execute*`()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwf_execute_dft_c2r*`()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of the FFT problem to be solved

Constraint: rank >= 1

`n` Input parameter

`n` is `const int`

`n` is an array of dimension `[rank]`.

`n` contains the size of each dimension of the FFT to be solved

Constraint: `n[i]` >= 1 for i in 1, rank

`howmany` Input parameter

`howmany` is `int`

`howmany` is the number of multi-dimensional FFT transforms to be computed

Constraint: `howmany` >= 1

in Input parameter

in is fftwf_complex*

in is a pointer to a previously allocated array of fftwf_complex data. At fftwf_execute() time this will have been filled with the input sequence on which to perform the FFT. The input data for set k starts at address in+k*idist.

Storage: Data in arrays will be stored in row-major order.

inembed Input parameter

inembed is const int

inembed is an array of dimension [rank] or NULL. This array denotes the dimensionality of a potentially larger space of which the input array, in, dimensions given by n, is a sub array.

Constraint: Either inembed[i] >= n[i] for i in 0, rank-1 OR if inembed == NULL, then inembed is assumed to have the same dimensions as n

istride Input parameter

istride is int

istride is the distance between successive elements of the input data

idist Input parameter

idist is int

idist is the distance between the start of each of the howmany transforms in the input data

out Input parameter

out is float

out is a pointer to a previously allocated array of float data. At fftwf_execute() time this will be filled with the resulting transform coefficients The output data for set k starts at address out+k*odist.

Storage: Data in arrays will be stored in row-major order.

onembed Input parameter

onembed is const int

onembed is an array of dimension [rank] or NULL. This array denotes the dimensionality of a potentially larger space of which the input array, out, dimensions given by n, is a sub array.

Constraint: Either onembed[i] >= n[i] for i in 0, rank-1 OR if onembed == NULL, then onembed is assumed to have the same dimensions as n

ostride Input parameter

ostride is int

ostride is the distance between successive elements of the output data

odist Input parameter

odist is int

odist is the distance between the start of each of the howmany transforms in the output data

flags Input parameter

flags is unsigned

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in flags are:

- FFTW_ESTIMATE – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.

- FFTW_MEASURE – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- FFTW_PATIENT – Produce a plan, testing more options than FFTW_MEASURE. Input and output data might be overwritten.
- FFTW_EXHAUSTIVE – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- FFTW_WISDOM_ONLY – Produce a plan only if the requested case has existing wisdom, otherwise return NULL.
- FFTW_PRESERVE_INPUT – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- FFTW_DESTROY_INPUT – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- FFTW_CONSERVE_MEMORY – Request that extra memory is not created. This might be ignored.
- FFTW_UNALIGNED – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by FFTW_MEASURE, FFTW_PATIENT and FFTW_EXHAUSTIVE will be limited when using *fftwf_set_timelimit*.
- If multiple options for planning rigor are included in flags, then the highest level of planning requested will be used.

5.5.19 fftwh_plan_dft_c2r

fftwh_plan_dft_c2r is a basic FFTW interface call for planning an *n*-dimensional complex-to-real FFT operation on a single, contiguous data sequence.

Note: in and out can be the same pointer for an "in-place" transform.

Syntax

C specification:

```
#include "fftw3.h"

fftwh_plan fftwh_plan_dft_c2r(int rank, const int *n, fftwh_complex *in,
                              __fp16 *out, unsigned flags);
```

Returns

This function returns an fftwh_plan object that can be used in subsequent *fftwh_execute*() calls on the previously supplied in and out arrays, or using different arrays via *fftwh_execute_dft_c2r*().

Parameters

rank Input parameter

rank is int

The number of dimensions of the FFT problem to be solved

Constraint: `rank` >= 1

`n` Input parameter

`n` is `const int*`.

`n` is an array of dimension `[rank]`.

`n` contains the size of each dimension of the FFT to be solved

Constraint: `n[i]` >= 1 for i in 1, `rank`

`in` Input parameter

in is fftwh_complex

`in` is a pointer to a previously allocated array of `fftwh_complex` data. At `fftwh_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `__fp16`

`out` is a pointer to a previously allocated array of `__fp16` data. At `fftwh_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwh_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.5.20 fftwh_plan_dft_c2r_1d

`fftwh_plan_dft_c2r_1d` is a basic FFTW interface call for planning a 1-d complex-to-real FFT operation on a single, contiguous data sequence.

Note: `in` and `out` can be the same pointer for an "in-place" transform.

Syntax

C specification:

```
#include "fftw3.h"

fftwh_plan fftwh_plan_dft_c2r_1d(int n0, fftwh_complex *in, __fp16 *out,
                                 unsigned flags);
```

Returns

This function returns an `fftwh_plan` object that can be used in subsequent *fftwh_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwh_execute_dft_c2r* `()`.

Parameters

`n0` Input parameter

`n0` is `int`

This is the length of the FFT sequence

Constraint: `n0` >= 1

`in` Input parameter

`in` is `fftwh_complex*`

`in` is a pointer to a previously allocated array of `fftwh_complex` data. At `fftwh_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `__fp16`

`out` is a pointer to a previously allocated array of `__fp16` data. At `fftwh_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.

- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwh_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.5.21 fftwh_plan_dft_c2r_2d

`fftwh_plan_dft_c2r_2d` is a basic FFTW interface call for planning a 2-d complex-to-real FFT operation on a single, contiguous data sequence.

Note: `in` and `out` can be the same pointer for an "in-place" transform.

Syntax

C specification:

```
#include "fftw3.h"

fftwh_plan fftwh_plan_dft_c2r_2d(int n0, int n1, fftwh_complex *in,
                                 __fp16 *out, unsigned flags);
```

Returns

This function returns an `fftwh_plan` object that can be used in subsequent *fftwh_execute*`()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwh_execute_dft_c2r*`()`.

Parameters

n0 Input parameter

`n0` is `int`

This is the length of the first dimension of the FFT sequence

Constraint: `n0` >= 1

n1 Input parameter

n1 is int

This is the length of the second dimension of the FFT sequence

Constraint: n1 >= 1

in Input parameter

in is fftwh_complex*

in is a pointer to a previously allocated array of fftwh_complex data. At fftwh_execute() time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

out Input parameter

out is __fp16

out is a pointer to a previously allocated array of __fp16 data. At fftwh_execute() time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

flags Input parameter

flags is unsigned

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in flags are:

- FFTW_ESTIMATE – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- FFTW_MEASURE – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- FFTW_PATIENT – Produce a plan, testing more options than FFTW_MEASURE. Input and output data might be overwritten.
- FFTW_EXHAUSTIVE – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- FFTW_WISDOM_ONLY – Produce a plan only if the requested case has existing wisdom, otherwise return NULL.
- FFTW_PRESERVE_INPUT – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- FFTW_DESTROY_INPUT – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- FFTW_CONSERVE_MEMORY – Request that extra memory is not created. This might be ignored.
- FFTW_UNALIGNED – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by FFTW_MEASURE, FFTW_PATIENT and FFTW_EXHAUSTIVE will be limited when using *fftwh_set_timelimit*.
- If multiple options for planning rigor are included in flags, then the highest level of planning requested will be used.

5.5.22 fftwh_plan_dft_c2r_3d

`fftwh_plan_dft_c2r_3d` is a basic FFTW interface call for planning a 3-d complex-to-real FFT operation on a single, contiguous data sequence.

Note: `in` and `out` can be the same pointer for an "in-place" transform.

Syntax

C specification:

```
#include "fftw3.h"

fftwh_plan fftwh_plan_dft_c2r_3d(int n0, int n1, int n2, fftwh_complex *in,
                                 __fp16 *out, unsigned flags);
```

Returns

This function returns an `fftwh_plan` object that can be used in subsequent *fftwh_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwh_execute_dft_c2r* `()`.

Parameters

`n0` Input parameter

`n0` is `int`

This is the length of the first dimension of the FFT sequence

Constraint: `n0` >= 1

`n1` Input parameter

`n1` is `int`

This is the length of the second dimension of the FFT sequence

Constraint: `n1` >= 1

`n2` Input parameter

`n2` is `int`

This is the length of the third dimension of the FFT sequence

Constraint: `n2` >= 1

`in` Input parameter

`in` is fftwh_complex

`in` is a pointer to a previously allocated array of `fftwh_complex` data. At `fftwh_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `__fp16`

`out` is a pointer to a previously allocated array of `__fp16` data. At `fftwh_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

flags Input parameter

flags is unsigned

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in flags are:

- FFTW_ESTIMATE – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- FFTW_MEASURE – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- FFTW_PATIENT – Produce a plan, testing more options than FFTW_MEASURE. Input and output data might be overwritten.
- FFTW_EXHAUSTIVE – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- FFTW_WISDOM_ONLY – Produce a plan only if the requested case has existing wisdom, otherwise return NULL.
- FFTW_PRESERVE_INPUT – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- FFTW_DESTROY_INPUT – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- FFTW_CONSERVE_MEMORY – Request that extra memory is not created. This might be ignored.
- FFTW_UNALIGNED – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by FFTW_MEASURE, FFTW_PATIENT and FFTW_EXHAUSTIVE will be limited when using *fftwh_set_timelimit*.
- If multiple options for planning rigor are included in flags, then the highest level of planning requested will be used.

5.5.23 fftwh_plan_guru_dft_c2r

fftwh_plan_guru_dft_c2r is a guru FFTW interface call for planning an *n*-dimensional complex-to-real FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner.

Syntax

C specification:

```
#include "fftw3.h"

fftwh_plan fftwh_plan_guru_dft_c2r(int rank, const fftwh_iodim *dims,
                                   int howmany_rank,
                                   const fftwh_iodim *howmany_dims,
                                   fftwh_complex *in, __fp16 *out,
                                   unsigned flags);
```

Returns

This function returns an `fftwh_plan` object that can be used in subsequent *fftwh_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwh_execute_dft* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

`dims` Input parameter

`dims` is `const fftwh_iodim *`

`dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the length of the `i` th dimension.
- `int is` – the stride between successive elements in the input vector in the `i` th dimension.
- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: `dims.n[i]` >= 1 for i in 0, `rank-1`

`howmany_rank` Input parameter

`howmany_rank` is `int`

`howmany_rank` is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then `howmany_rank` needs only be 1.

Constraint: `howmany` >= 0

Note: If `howmany_rank=0`, then a single FFT transform is computed.

`howmany_dims` Input parameter

`howmany_dims` is `const fftwh_iodim *`

`howmany_dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the number of transforms to be computed of the `i` th dimension.of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `int is` – the number of memory elements, of type `fftwh_complex`, between the start of transforms in the `i` th dimension. of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the `howmany_dims.n[]` entries. For example, if `howmany_rank=1`, then `howmany_dims[0].n` is the memory distance between successive transforms.

Constraint: `howmany_dims.n[i]` >= 1 for i in 0, `howmany_rank-1`

in Input parameter

in is fftwh_complex*

in is a pointer to a previously allocated array of fftwh_complex data. At fftwh_execute() time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

out Input parameter

out is fftwh_complex*

out is a pointer to a previously allocated array of __fp16 data. At fftwh_execute() time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

flags Input parameter

flags is unsigned

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in flags are:

- FFTW_ESTIMATE – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- FFTW_MEASURE – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- FFTW_PATIENT – Produce a plan, testing more options than FFTW_MEASURE. Input and output data might be overwritten.
- FFTW_EXHAUSTIVE – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- FFTW_WISDOM_ONLY – Produce a plan only if the requested case has existing wisdom, otherwise return NULL.
- FFTW_PRESERVE_INPUT – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- FFTW_DESTROY_INPUT – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- FFTW_CONSERVE_MEMORY – Request that extra memory is not created. This might be ignored.
- FFTW_UNALIGNED – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by FFTW_MEASURE, FFTW_PATIENT and FFTW_EXHAUSTIVE will be limited when using *fftwh_set_timelimit*.
- If multiple options for planning rigor are included in flags, then the highest level of planning requested will be used.

5.5.24 fftwh_plan_guru_split_dft_c2r

fftwh_plan_guru_split_dft_c2r is a guru FFTW interface call for planning an *n*-dimensional complex-to-real FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner. Input and output data is stored in separate arrays for real and imaginary parts.

Syntax

C specification:

```
#include "fftw3.h"

fftwh_plan fftwh_plan_guru_split_dft_c2r(int rank, const fftwh_iodim *dims,
                                         int howmany_rank,
                                         const fftwh_iodim *howmany_dims,
                                         __fp16 *ri, __fp16 *ii, __fp16 *out,
                                         unsigned flags);
```

Returns

This function returns an `fftwh_plan` object that can be used in subsequent *fftwh_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwh_execute_dft* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

`dims` Input parameter

`dims` is `const fftwh_iodim *`

`dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the length of the `i` th dimension.
- `int is` – the stride between successive elements in the input vector in the `i` th dimension.
- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: `dims.n[i]` >= 1 for i in 0, `rank-1`

`howmany_rank` Input parameter

`howmany_rank` is `int`

`howmany_rank` is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then `howmany_rank` needs only be 1.

Constraint: `howmany` >= 0

Note: If `howmany_rank=0`, then a single FFT transform is computed.

`howmany_dims` Input parameter

`howmany_dims` is `const fftwh_iodim *`

`howmany_dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the number of transforms to be computed of the `i` th dimension.of the `howmany_rank`-dimensional space mapped by the transforms to be computed.

- `int is` – the number of memory elements, of type `fftwh_complex`, between the start of transforms in the `i` th dimension. of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the `howmany_dims.n[]` entries. For example, if `howmany_rank=1`, then `howmany_dims[0].n` is the memory distance between successive transforms.

Constraint: `howmany_dims.n[i]` >= 1 for i in 0, `howmany_rank-1`

`ri` Input parameter

`ri` is `__fp16 *`

`ri` is a pointer to a previously allocated array of `__fp16` data. At `fftwh_execute()` time this will have been filled with the real parts of the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`ii` Input parameter

`ii` is `__fp16 *`

`ii` is a pointer to a previously allocated array of `__fp16` data. At `fftwh_execute()` time this will have been filled with the imaginary parts of the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `__fp16 *`

`out` is a pointer to a previously allocated array of `__fp16` data. At `fftwh_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.

- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwh_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.5.25 fftwh_plan_guru64_dft_c2r

`fftwh_plan_guru64_dft_c2r` is a guru FFTW interface call for planning an *n*-dimensional complex-to-real FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner.

Unlike *fftwh_plan_guru_dft_c2r*, `fftwh_plan_guru64_dft_c2r` allows selected parameters to be specified as 64-bit integers. For Arm Performance Libraries, this parameter size difference is true in both `armpl*_lp64` and `armpl_ilp64` variants. Therefore, the parameter size difference is independent of the integer size used in the rest of the library interface.

Syntax

C specification:

```
#include "fftw3.h"

fftwh_plan fftwh_plan_guru64_dft_c2r(int rank, const fftwh_iodim64 *dims,
                                     int howmany_rank,
                                     const fftwh_iodim64 *howmany_dims,
                                     fftwh_complex *in, __fp16 *out,
                                     unsigned flags);
```

Returns

This function returns an `fftwh_plan` object that can be used in subsequent *fftwh_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwh_execute_dft* `()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

`dims` Input parameter

`dims` is `const fftwh_iodim64 *`

`dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `ptrdiff_t n` – the length of the `i` th dimension.
- `ptrdiff_t is` – the stride between successive elements in the input vector in the `i` th dimension.
- `ptrdiff_t os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: `dims.n[i]` >= 1 for i in 0, `rank-1`

`howmany_rank` Input parameter

`howmany_rank` is `int`

`howmany_rank` is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then `howmany_rank` needs only be 1.

Constraint: `howmany` >= 0

Note: If `howmany_rank=0`, then a single FFT transform is computed.

`howmany_dims` Input parameter

`howmany_dims` is `const fftwh_iodim64 *`

`howmany_dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `ptrdiff_t n` – the number of transforms to be computed of the `i` th dimension.of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `ptrdiff_t is` – the number of memory elements, of type `fftwh_complex`, between the start of transforms in the `i` th dimension. of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `ptrdiff_t os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the `howmany_dims.n[]` entries. For example, if `howmany_rank=1`, then `howmany_dims[0].n` is the memory distance between successive transforms.

Constraint: `howmany_dims.n[i]` >= 1 for i in 0, `howmany_rank-1`

`in` Input parameter

`in` is `fftwh_complex*`

`in` is a pointer to a previously allocated array of `fftwh_complex` data. At `fftwh_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `__fp16`

`out` is a pointer to a previously allocated array of `__fp16` data. At `fftwh_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.

- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwh_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.5.26 fftwh_plan_guru64_split_dft_c2r

`fftwh_plan_guru64_split_dft_c2r` is a guru FFTW interface call for planning an n-dimensional complex-to-real FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner. Input and output data is stored in separate arrays for real and imaginary parts.

Unlike *fftwh_plan_guru_split_dft_c2r*, `fftwh_plan_guru64_split_dft_c2r` allows selected parameters to be specified as 64-bit integers. For Arm Performance Libraries, this parameter size difference is true in both `armpl*_lp64` and `armpl_ilp64` variants. Therefore, the parameter size difference is independent of the integer size used in the rest of the library interface.

Syntax

C specification:

```
#include "fftw3.h"

fftwh_plan fftwh_plan_guru_split_dft_c2r(int rank, const fftwh_iodim64 *dims,
                                         int howmany_rank,
                                         const fftwh_iodim64 *howmany_dims,
                                         __fp16 *ri, __fp16 *ii, __fp16 *out,
                                         unsigned flags);
```

Returns

This function returns an `fftwh_plan` object that can be used in subsequent *fftwh_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwh_execute_dft* `()`.

Parameters

rank Input parameter

rank is int

The number of dimensions of each individual FFT problem to be solved.

Constraint: rank >= 1

dims Input parameter

dims is const fftwh_iodim64 *

dims is an array of structures, dimension [rank], where the members of dims[i], for i in 0, rank-1, are defined to be:

- ptrdiff_t n – the length of the i th dimension.
- ptrdiff_t is – the stride between successive elements in the input vector in the i th dimension.
- ptrdiff_t os – the stride between successive elements in the output vector in the i th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: dims.n[i] >= 1 for i in 0, rank-1

howmany_rank Input parameter

howmany_rank is int

howmany_rank is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then howmany_rank needs only be 1.

Constraint: howmany >= 0

Note: If howmany_rank=0, then a single FFT transform is computed.

howmany_dims Input parameter

howmany_dims is const fftwh_iodim64 *

howmany_dims is an array of structures, dimension [rank], where the members of dims[i], for i in 0, rank-1, are defined to be:

- ptrdiff_t n – the number of transforms to be computed of the i th dimension.of the howmany_rank-dimensional space mapped by the transforms to be computed.
- ptrdiff_t is – the number of memory elements, of type fftwh_complex, between the start of transforms in the i th dimension. of the howmany_rank-dimensional space mapped by the transforms to be computed.
- ptrdiff_t os – the stride between successive elements in the output vector in the i th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the howmany_dims.n[] entries. For example, if howmany_rank=1, then howmany_dims[0].n is the memory distance between successive transforms.

Constraint: howmany_dims.n[i] >= 1 for i in 0, howmany_rank-1

ri Input parameter

ri is __fp16 *

ri is a pointer to a previously allocated array of __fp16 data. At fftwh_execute() time this will have been filled with the real parts of the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

ii Input parameter

ii is __fp16 *

ii is a pointer to a previously allocated array of __fp16 data. At fftwh_execute() time this will have been filled with the imaginary parts of the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

out Input parameter

out is __fp16 *

out is a pointer to a previously allocated array of __fp16 data. At fftwh_execute() time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

flags Input parameter

flags is unsigned

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in flags are:

- FFTW_ESTIMATE – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- FFTW_MEASURE – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- FFTW_PATIENT – Produce a plan, testing more options than FFTW_MEASURE. Input and output data might be overwritten.
- FFTW_EXHAUSTIVE – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- FFTW_WISDOM_ONLY – Produce a plan only if the requested case has existing wisdom, otherwise return NULL.
- FFTW_PRESERVE_INPUT – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- FFTW_DESTROY_INPUT – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- FFTW_CONSERVE_MEMORY – Request that extra memory is not created. This might be ignored.
- FFTW_UNALIGNED – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by FFTW_MEASURE, FFTW_PATIENT and FFTW_EXHAUSTIVE will be limited when using *fftwh_set_timelimit*.
- If multiple options for planning rigor are included in flags, then the highest level of planning requested will be used.

5.5.27 fftwh_plan_many_dft_c2r

fftwh_plan_many_dft_c2r is an advanced FFTW interface call for planning an *n*-dimensional complex-to-real FFT operation on howmany sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner.

For more information on the use of advanced planning calls see *How to choose which FFT routine you need*.

Syntax

C specification:

```
#include "fftw3.h"

fftwh_plan fftwh_plan_many_dft_c2r(int rank, const int *n, int howmany,
                                   fftwh_complex *in, const int *inembed,
                                   int istride, int idist, __fp16 *out,
                                   const int *onembed, int ostride,
                                   int odist, unsigned flags);
```

Returns

This function returns an `fftwh_plan` object that can be used in subsequent *fftwh_execute*`()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwh_execute_dft_c2r*`()`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of the FFT problem to be solved

Constraint: rank >= 1

`n` Input parameter

`n` is `const int`

`n` is an array of dimension `[rank]`.

`n` contains the size of each dimension of the FFT to be solved

Constraint: `n[i]` >= 1 for i in 1, rank

`howmany` Input parameter

`howmany` is `int`

`howmany` is the number of multi-dimensional FFT transforms to be computed

Constraint: `howmany` >= 1

`in` Input parameter

`in` is `fftwh_complex*`

`in` is a pointer to a previously allocated array of `fftwh_complex` data. At `fftwh_execute()` time this will have been filled with the input sequence on which to perform the FFT. The input data for set `k` starts at address `in+k*idist`.

Storage: Data in arrays will be stored in row-major order.

`inembed` Input parameter

`inembed` is `const int`

`inembed` is an array of dimension `[rank]` or `NULL`. This array denotes the dimensionality of a potentially larger space of which the input array, `in`, dimensions given by `n`, is a sub array.

Constraint: Either `inembed[i]` >= `n[i]` for `i` in 0, `rank-1` OR if `inembed` == `NULL`, then `inembed` is assumed to have the same dimensions as `n`

istride Input parameter

istride is int

istride is the distance between successive elements of the input data

idist Input parameter

idist is int

idist is the distance between the start of each of the howmany transforms in the input data

out Input parameter

out is __fp16

out is a pointer to a previously allocated array of __fp16 data. At fftwh_execute() time this will be filled with the resulting transform coefficients The output data for set k starts at address out+k*odist.

Storage: Data in arrays will be stored in row-major order.

onembed Input parameter

onembed is const int

onembed is an array of dimension [rank] or NULL. This array denotes the dimensionality of a potentially larger space of which the input array, out, dimensions given by n, is a sub array.

Constraint: Either onembed[i] >= n[i] for i in 0, rank-1 OR if onembed == NULL, then onembed is assumed to have the same dimensions as n

ostride Input parameter

ostride is int

ostride is the distance between successive elements of the output data

odist Input parameter

odist is int

odist is the distance between the start of each of the howmany transforms in the output data

flags Input parameter

flags is unsigned

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in flags are:

- FFTW_ESTIMATE – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- FFTW_MEASURE – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- FFTW_PATIENT – Produce a plan, testing more options than FFTW_MEASURE. Input and output data might be overwritten.
- FFTW_EXHAUSTIVE – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- FFTW_WISDOM_ONLY – Produce a plan only if the requested case has existing wisdom, otherwise return NULL.
- FFTW_PRESERVE_INPUT – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- FFTW_DESTROY_INPUT – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- FFTW_CONSERVE_MEMORY – Request that extra memory is not created. This might be ignored.

- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwh_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.6 FFT real to real functions

5.6.1 fftw_plan_guru_r2r

`fftw_plan_many_r2r` is a guru FFTW interface call for planning an *n*-dimensional complex-to-complex FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return `NULL`.

Syntax

C specification:

```
#include "fftw3.h"

fftw_plan fftw_plan_guru_r2r(int rank, const fftw_iodim *dims,
                             int howmany_rank,
                             const fftw_iodim *howmany_dims,
                             double *in, double *out,
                             unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_plan_guru_r2r(rank,dims,howmany_rank, &
    howmany_dims,in,out,kind,flags) &
    bind(C, name='fftw_plan_guru_r2r')
  integer(C_INT), value :: rank
  type(fftw_iodim), dimension(*), intent(in) :: dims
  integer(C_INT), value :: howmany_rank
  type(fftw_iodim), dimension(*), intent(in) :: howmany_dims
  real(C_DOUBLE), dimension(*), intent(out) :: in
  real(C_DOUBLE), dimension(*), intent(out) :: out
  integer(C_FFTW_R2R_KIND), dimension(*), intent(in) :: kind
  integer(C_INT), value :: flags
end function fftw_plan_guru_r2r
```

Returns

An `fftw_plan` object is returned that can be used in subsequent *fftw_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftw_execute_r2r* `()`.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return `NULL`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

`dims` Input parameter

`dims` is `const fftw_iodim *`

`dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the length of the `i` th dimension.
- `int is` – the stride between successive elements in the input vector in the `i` th dimension.
- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: `dims.n[i]` >= 1 for i in 0, `rank-1`

`howmany_rank` Input parameter

`howmany_rank` is `int`

`howmany_rank` is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then `howmany_rank` needs only be 1.

Constraint: `howmany` >= 0

Note: If `howmany_rank=0`, then a single FFT transform is computed.

`howmany_dims` Input parameter

`howmany_dims` is `const fftw_iodim *`

`howmany_dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the number of transforms to be computed of the `i` th dimension.of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `int is` – the number of memory elements, of type `double`, between the start of transforms in the `i` th dimension. of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the `howmany_dims.n[]` entries. For example, if `howmany_rank=1`, then `howmany_dims[0].n` is the memory distance between successive transforms.

Constraint: `howmany_dims.n[i]` >= 1 for i in 0, `howmany_rank-1`

in Input parameter

`in` is `double*`

`in` is a pointer to a previously allocated array of `double` data. At `fftw_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

out Input parameter

`out` is `double*`

`out` is a pointer to a previously allocated array of `double` data. At `fftw_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

flags Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftw_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.6.2 fftw_plan_guru64_dft_r2r

`fftw_plan_guru64_dft` is a guru FFTW interface call for planning an *n*-dimensional real-to-real FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner.

Unlike *fftw_plan_guru_dft*, `fftw_plan_guru64_dft` allows selected parameters to be specified as 64-bit integers. For Arm Performance Libraries, this parameter size difference is true in both `armpl*_lp64` and `armpl_ilp64` variants. Therefore, the parameter size difference is independent of the integer size used in the rest of the library interface.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return `NULL`.

Syntax

C specification:

```
#include "fftw3.h"

fftw_plan fftw_plan_guru64_dft(int rank, const fftw_iodim64 *dims,
                               int howmany_rank,
                               const fftw_iodim64 *howmany_dims,
                               double *in, double *out,
                               unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_plan_guru64_dft(rank,dims,howmany_rank, &
    howmany_dims,in,out,sign,flags) &
    bind(C, name='fftw_plan_guru64_dft')
  integer(C_INT), value :: rank
  type(fftw_iodim64), dimension(*), intent(in) :: dims
  integer(C_INT), value :: howmany_rank
  type(fftw_iodim64), dimension(*), intent(in) :: howmany_dims
  complex(C_DOUBLE_COMPLEX), dimension(*), intent(out) :: in
  complex(C_DOUBLE_COMPLEX), dimension(*), intent(out) :: out
  integer(C_INT), value :: sign
  integer(C_INT), value :: flags
end function fftw_plan_guru64_dft
```

Returns

An `fftw_plan` object is returned that can be used in subsequent *fftw_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftw_execute_dft* `()`.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return `NULL`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

dims Input parameter

dims is const fftw_iodim64 *

dims is an array of structures, dimension [rank], where the members of dims[i], for i in 0, rank-1, are defined to be:

- ptrdiff_t n – the length of the i th dimension.
- ptrdiff_t is – the stride between successive elements in the input vector in the i th dimension.
- ptrdiff_t os – the stride between successive elements in the output vector in the i th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: dims.n[i] >= 1 for i in 0, rank-1

howmany_rank Input parameter

howmany_rank is int

howmany_rank is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then howmany_rank needs only be 1.

Constraint: howmany >= 0

Note: If howmany_rank=0, then a single FFT transform is computed.

howmany_dims Input parameter

howmany_dims is const fftw_iodim64 *

howmany_dims is an array of structures, dimension [rank], where the members of dims[i], for i in 0, rank-1, are defined to be:

- ptrdiff_t n – the number of transforms to be computed of the i th dimension.of the howmany_rank-dimensional space mapped by the transforms to be computed.
- ptrdiff_t is – the number of memory elements, of type double, between the start of transforms in the i th dimension. of the howmany_rank-dimensional space mapped by the transforms to be computed.
- ptrdiff_t os – the stride between successive elements in the output vector in the i th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the howmany_dims.n[] entries. For example, if howmany_rank=1, then howmany_dims[0].n is the memory distance between successive transforms.

Constraint: howmany_dims.n[i] >= 1 for i in 0, howmany_rank-1

in Input parameter

in is double*

in is a pointer to a previously allocated array of double data. At fftw_execute() time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

out Input parameter

out is double*

out is a pointer to a previously allocated array of double data. At fftw_execute() time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

flags Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftw_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.6.3 fftw_plan_many_r2r

`fftw_plan_many_r2r` is an advanced FFTW interface call for planning an *n*-dimensional real-to-real FFT operation on `howmany` sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return `NULL`.

Syntax

C specification:

```
#include "fftw3.h"

fftw_plan fftw_plan_many_r2r(int rank, const int *n, int howmany,
                             double *in, const int *inembed, int istride,
                             int idist, double *out, const int *onembed,
                             int ostride, int odist, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_plan_many_r2r(rank,n,howmany,in,inembed, &
    istride,idist,out,onembed,ostride,odist,kind,flags) &
    bind(C, name='fftw_plan_many_r2r')
  integer(C_INT), value :: rank
  integer(C_INT), dimension(*), intent(in) :: n
  integer(C_INT), value :: howmany
  real(C_DOUBLE), dimension(*), intent(out) :: in
  integer(C_INT), dimension(*), intent(in) :: inembed
  integer(C_INT), value :: istride
  integer(C_INT), value :: idist
  real(C_DOUBLE), dimension(*), intent(out) :: out
  integer(C_INT), dimension(*), intent(in) :: onembed
  integer(C_INT), value :: ostride
  integer(C_INT), value :: odist
  integer(C_FFTW_R2R_KIND), dimension(*), intent(in) :: kind
  integer(C_INT), value :: flags
end function fftw_plan_many_r2r
```

Returns

An `fftw_plan` object is returned that can be used in subsequent *fftw_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftw_execute_r2r* `()`.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return `NULL`.

Parameters

`rank`

Input parameter

`rank` is `int`

The number of dimensions of the FFT problem to be solved

Constraint: rank >= 1

`n` Input parameter

`n` is `const int`

`n` is an array of dimension `[rank]`.

`n` contains the size of each dimension of the FFT to be solved

Constraint: `n[i]` >= 1 for i in 1, rank

`howmany` Input parameter

`howmany` is `int`

`howmany` is the number of multi-dimensional FFT transforms to be computed

Constraint: `howmany` >= 1

`in` Input parameter

`in` is `double*`

`in` is a pointer to a previously allocated array of `double` data. At `fftw_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`inembed` Input parameter

`inembed` is `const int`

`inembed` is a pointer to a previously allocated array of `double` data. This array denotes the dimensionality of a potentially larger space of which the input array, `in`, dimensions given by `n`, is a sub array.

Constraint: Either `inembed[i] >= n[i]` for `i` in 0, `rank-1` OR if `inembed == NULL`, then `inembed` is assumed to have the same dimensions as `n`

`istride` Input parameter

`istride` is `int`

`istride` is the distance between successive elements of the input data

`idist` Input parameter

`idist` is `int`

`idist` is the distance between the start of each of the `howmany` transforms in the input data

`out`

Input parameter

`out` is `double*`

`out` is a pointer to a previously allocated array of `double` data. At `fftw_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`onembed` Input parameter

`onembed` is `const int`

`onembed` is an array of dimension `[rank]` or `NULL`. This array denotes the dimensionality of a potentially larger space of which the input array, `out`, dimensions given by `n`, is a sub array.

Constraint: Either `onembed[i] >= n[i]` for `i` in 0, `rank-1` OR if `onembed == NULL`, then `onembed` is assumed to have the same dimensions as `n`

`ostride` Input parameter

`ostride` is `int`

`ostride` is the distance between successive elements of the output data

`odist` Input parameter

`odist` is `int`

`odist` is the distance between the start of each of the `howmany` transforms in the output data

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.

- FFTW_PATIENT – Produce a plan, testing more options than FFTW_MEASURE. Input and output data might be overwritten.
- FFTW_EXHAUSTIVE – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- FFTW_WISDOM_ONLY – Produce a plan only if the requested case has existing wisdom, otherwise return NULL.
- FFTW_PRESERVE_INPUT – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- FFTW_DESTROY_INPUT – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- FFTW_CONSERVE_MEMORY – Request that extra memory is not created. This might be ignored.
- FFTW_UNALIGNED – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by FFTW_MEASURE, FFTW_PATIENT and FFTW_EXHAUSTIVE will be limited when using *fftw_set_timelimit*.
- If multiple options for planning rigor are included in flags, then the highest level of planning requested will be used.

5.6.4 fftw_plan_r2r

fftw_plan_r2r is a basic FFTW interface call for planning an *n*-dimensional real-to-real FFT operation on a single, contiguous data sequence.

Note: in and out can be the same pointer for an "in-place" transform.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return NULL.

Syntax

C specification:

```
#include "fftw3.h"

fftw_plan fftw_plan_r2r(int rank, const int *n, double *in, double *out,
                        unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_plan_r2r(rank,n,in,out,kind,flags) &
    bind(C, name='fftw_plan_r2r')
  integer(C_INT), value :: rank
  integer(C_INT), dimension(*), intent(in) :: n
  real(C_DOUBLE), dimension(*), intent(out) :: in
  real(C_DOUBLE), dimension(*), intent(out) :: out
```

(continues on next page)

(continued from previous page)

```
  integer(C_FFTW_R2R_KIND), dimension(*), intent(in) :: kind
  integer(C_INT), value :: flags
end function fftw_plan_r2r
```

Returns

An `fftw_plan` object is returned that can be used in subsequent *fftw_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftw_execute_r2r* `()`.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return `NULL`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of the FFT problem to be solved

Constraint: `rank` >= 1

`n` Input parameter

`n` is `const int*`.

`n` is an array of dimension `[rank]`.

`n` contains the size of each dimension of the FFT to be solved

Constraint: `n[i]` >=1 for i in 1, `rank`

`in` Input parameter

`in` is `double*`

`in` is a pointer to a previously allocated array of `double` data. At `fftw_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `double*`

`out` is a pointer to a previously allocated array of `double` data. At `fftw_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.

- FFTW_EXHAUSTIVE – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- FFTW_WISDOM_ONLY – Produce a plan only if the requested case has existing wisdom, otherwise return NULL.
- FFTW_PRESERVE_INPUT – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- FFTW_DESTROY_INPUT – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- FFTW_CONSERVE_MEMORY – Request that extra memory is not created. This might be ignored.
- FFTW_UNALIGNED – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by FFTW_MEASURE, FFTW_PATIENT and FFTW_EXHAUSTIVE will be limited when using *fftw_set_timelimit*.
- If multiple options for planning rigor are included in flags, then the highest level of planning requested will be used.

5.6.5 fftw_plan_r2r_1d

fftw_plan_r2r_1d is a basic FFTW interface call for planning a 1-d real-to-real FFT operation on a single, contiguous data sequence.

Note: in and out can be the same pointer for an "in-place" transform.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return NULL.

Syntax

C specification:

```
#include "fftw3.h"

fftw_plan fftw_plan_r2r_1d(int n0, double *in, double *out, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_plan_r2r_1d(n,in,out,kind,flags) &
    bind(C, name='fftw_plan_r2r_1d')
  integer(C_INT), value :: n
  real(C_DOUBLE), dimension(*), intent(out) :: in
  real(C_DOUBLE), dimension(*), intent(out) :: out
  integer(C_FFTW_R2R_KIND), value :: kind
  integer(C_INT), value :: flags
end function fftw_plan_r2r_1d
```

Returns

An `fftw_plan` object is returned that can be used in subsequent *fftw_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftw_execute_r2r* `()`.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return `NULL`.

Parameters

`n0` Input parameter

`n0` is `int`

This is the length of the FFT sequence

Constraint: `n0` >= 1

`in` Input parameter

`in` is `double*`

`in` is a pointer to a previously allocated array of `double` data. At `fftw_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `double*`

`out` is a pointer to a previously allocated array of `double` data. At `fftw_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.

- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftw_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.6.6 fftw_plan_r2r_2d

`fftw_plan_r2r_2d` is a basic FFTW interface call for planning a 2-d real-to-real FFT operation on a single, contiguous data sequence.

Note: `in` and `out` can be the same pointer for an "in-place" transform.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return `NULL`.

Syntax

C specification:

```
#include "fftw3.h"

fftw_plan fftw_plan_r2r_2d(int n0, int n1, double *in, double *out,
                           unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_plan_r2r_2d(n0,n1,in,out,kind0,kind1,flags) &
    bind(C, name='fftw_plan_r2r_2d')
  integer(C_INT), value :: n0
  integer(C_INT), value :: n1
  real(C_DOUBLE), dimension(*), intent(out) :: in
  real(C_DOUBLE), dimension(*), intent(out) :: out
  integer(C_FFTW_R2R_KIND), value :: kind0
  integer(C_FFTW_R2R_KIND), value :: kind1
  integer(C_INT), value :: flags
end function fftw_plan_r2r_2d
```

Returns

An `fftw_plan` object is returned that can be used in subsequent *fftw_execute*`()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftw_execute_r2r*`()`.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return `NULL`.

Parameters

n0 Input parameter

`n0` is `int`

This is the length of the first dimension of the FFT sequence

Constraint: `n0` >= 1

n1 Input parameter

`n1` is `int`

This is the length of the second dimension of the FFT sequence

Constraint: `n1` >= 1

in Input parameter

`in` is `double*`

`in` is a pointer to a previously allocated array of `double` data. At `fftw_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

out Input parameter

`out` is `double*`

`out` is a pointer to a previously allocated array of `double` data. At `fftw_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

flags Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftw_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.6.7 fftw_plan_r2r_3d

`fftw_plan_r2r_3d` is a basic FFTW interface call for planning a 3-d real-to-real FFT operation on a single, contiguous data sequence.

Note: `in` and `out` can be the same pointer for an "in-place" transform.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return `NULL`.

Syntax

C specification:

```
#include "fftw3.h"

fftw_plan fftw_plan_r2r_3d(int n0, int n1, int n2, double *in, double *out,
                           unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_plan_r2r_3d(n0,n1,n2,in,out,kind0,kind1, &
    kind2,flags) bind(C, name='fftw_plan_r2r_3d')
  integer(C_INT), value :: n0
  integer(C_INT), value :: n1
  integer(C_INT), value :: n2
  real(C_DOUBLE), dimension(*), intent(out) :: in
  real(C_DOUBLE), dimension(*), intent(out) :: out
  integer(C_FFTW_R2R_KIND), value :: kind0
  integer(C_FFTW_R2R_KIND), value :: kind1
  integer(C_FFTW_R2R_KIND), value :: kind2
  integer(C_INT), value :: flags
end function fftw_plan_r2r_3d
```

Returns

An `fftw_plan` object is returned that can be used in subsequent *fftw_execute*`()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftw_execute_r2r*`()`.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return `NULL`.

Parameters

n0 Input parameter

`n0` is `int`

This is the length of the first dimension of the FFT sequence

Constraint: `n0` >= 1

n1 Input parameter

`n1` is `int`

This is the length of the second dimension of the FFT sequence

Constraint: `n1` >= 1

n2 Input parameter

`n2` is `int`

This is the length of the third dimension of the FFT sequence

Constraint: `n2` >= 1

in Input parameter

`in` is double

`in` is a pointer to a previously allocated array of `double` data. At `fftw_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

out Input parameter

`out` is `double*`

`out` is a pointer to a previously allocated array of `double` data. At `fftw_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

flags Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.

- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftw_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.6.8 fftwf_plan_guru_r2r

`fftwf_plan_many_r2r` is a guru FFTW interface call for planning an *n*-dimensional complex-to-complex FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return `NULL`.

Syntax

C specification:

```
#include "fftw3.h"

fftwf_plan fftwf_plan_guru_r2r(int rank, const fftwf_iodim *dims,
                               int howmany_rank,
                               const fftwf_iodim *howmany_dims,
                               float *in, float *out,
                               unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_plan_guru_r2r(rank,dims,howmany_rank, &
    howmany_dims,in,out,kind,flags) &
    bind(C, name='fftwf_plan_guru_r2r')
  integer(C_INT), value :: rank
  type(fftwf_iodim), dimension(*), intent(in) :: dims
  integer(C_INT), value :: howmany_rank
  type(fftwf_iodim), dimension(*), intent(in) :: howmany_dims
  real(C_FLOAT), dimension(*), intent(out) :: in
  real(C_FLOAT), dimension(*), intent(out) :: out
  integer(C_FFTW_R2R_KIND), dimension(*), intent(in) :: kind
  integer(C_INT), value :: flags
end function fftwf_plan_guru_r2r
```

Returns

This function returns an `fftwf_plan` object that can be used in subsequent *fftwf_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwf_execute_r2r* `()`.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return `NULL`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

`dims` Input parameter

`dims` is `const fftwf_iodim *`

`dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the length of the `i` th dimension.
- `int is` – the stride between successive elements in the input vector in the `i` th dimension.
- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: `dims.n[i]` >= 1 for i in 0, `rank-1`

`howmany_rank` Input parameter

`howmany_rank` is `int`

`howmany_rank` is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then `howmany_rank` needs only be 1.

Constraint: `howmany` >= 0

Note: If `howmany_rank=0`, then a single FFT transform is computed.

`howmany_dims` Input parameter

`howmany_dims` is `const fftwf_iodim *`

`howmany_dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the number of transforms to be computed of the `i` th dimension.of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `int is` – the number of memory elements, of type `float`, between the start of transforms in the `i` th dimension. of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the `howmany_dims.n[]` entries. For example, if `howmany_rank=1`, then `howmany_dims[0].n` is the memory distance between successive transforms.

Constraint: `howmany_dims.n[i]` >= 1 for i in 0, `howmany_rank-1`

in Input parameter

`in` is `float`

`in` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

out Input parameter

`out` is `float`

`out` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

flags Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwf_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.6.9 fftwf_plan_guru64_dft

`fftwf_plan_guru64_dft` is a guru FFTW interface call for planning an *n*-dimensional real-to-real FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner.

Unlike *fftwf_plan_guru_dft*, `fftwf_plan_guru64_dft` allows selected parameters to be specified as 64-bit integers. For Arm Performance Libraries, this parameter size difference is true in both `armpl*_lp64` and `armpl_ilp64` variants. Therefore, the parameter size difference is independent of the integer size used in the rest of the library interface.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return `NULL`.

Syntax

C specification:

```
#include "fftw3.h"

fftwf_plan fftwf_plan_guru64_dft(int rank, const fftwf_iodim64 *dims,
                                 int howmany_rank,
                                 const fftwf_iodim64 *howmany_dims,
                                 float *in, float *out,
                                 unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_plan_guru64_dft(rank,dims,howmany_rank, &
    howmany_dims,in,out,sign,flags) &
    bind(C, name='fftwf_plan_guru64_dft')
  integer(C_INT), value :: rank
  type(fftwf_iodim64), dimension(*), intent(in) :: dims
  integer(C_INT), value :: howmany_rank
  type(fftwf_iodim64), dimension(*), intent(in) :: howmany_dims
  complex(C_FLOAT_COMPLEX), dimension(*), intent(out) :: in
  complex(C_FLOAT_COMPLEX), dimension(*), intent(out) :: out
  integer(C_INT), value :: sign
  integer(C_INT), value :: flags
end function fftwf_plan_guru64_dft
```

Returns

This function returns an `fftwf_plan` object that can be used in subsequent *fftwf_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwf_execute_dft* `()`.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return `NULL`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

dims Input parameter

dims is const fftwf_iodim64 *

dims is an array of structures, dimension [rank], where the members of dims[i], for i in 0, rank-1, are defined to be:

- ptrdiff_t n – the length of the i th dimension.
- ptrdiff_t is – the stride between successive elements in the input vector in the i th dimension.
- ptrdiff_t os – the stride between successive elements in the output vector in the i th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: dims.n[i] >= 1 for i in 0, rank-1

howmany_rank Input parameter

howmany_rank is int

howmany_rank is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then howmany_rank needs only be 1.

Constraint: howmany >= 0

Note: If howmany_rank=0, then a single FFT transform is computed.

howmany_dims Input parameter

howmany_dims is const fftwf_iodim64 *

howmany_dims is an array of structures, dimension [rank], where the members of dims[i], for i in 0, rank-1, are defined to be:

- ptrdiff_t n – the number of transforms to be computed of the i th dimension.of the howmany_rank-dimensional space mapped by the transforms to be computed.
- ptrdiff_t is – the number of memory elements, of type float, between the start of transforms in the i th dimension. of the howmany_rank-dimensional space mapped by the transforms to be computed.
- ptrdiff_t os – the stride between successive elements in the output vector in the i th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the howmany_dims.n[] entries. For example, if howmany_rank=1, then howmany_dims[0].n is the memory distance between successive transforms.

Constraint: howmany_dims.n[i] >= 1 for i in 0, howmany_rank-1

in Input parameter

in is float

in is a pointer to a previously allocated array of float data. At fftwf_execute() time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

out Input parameter

out is float

out is a pointer to a previously allocated array of float data. At fftwf_execute() time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

flags Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwf_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.6.10 fftwf_plan_many_r2r

`fftwf_plan_many_r2r` is an advanced FFTW interface call for planning an *n*-dimensional real-to-real FFT operation on `howmany` sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return `NULL`.

Syntax

C specification:

```
#include "fftw3.h"

fftwf_plan fftwf_plan_many_r2r(int rank, const int *n, int howmany,
                               float *in, const int *inembed, int istride,
                               int idist, float *out, const int *onembed,
                               int ostride, int odist, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_plan_many_r2r(rank,n,howmany,in,inembed, &
    istride,idist,out,onembed,ostride,odist,kind,flags) &
    bind(C, name='fftwf_plan_many_r2r')
  integer(C_INT), value :: rank
  integer(C_INT), dimension(*), intent(in) :: n
  integer(C_INT), value :: howmany
  real(C_FLOAT), dimension(*), intent(out) :: in
  integer(C_INT), dimension(*), intent(in) :: inembed
  integer(C_INT), value :: istride
  integer(C_INT), value :: idist
  real(C_FLOAT), dimension(*), intent(out) :: out
  integer(C_INT), dimension(*), intent(in) :: onembed
  integer(C_INT), value :: ostride
  integer(C_INT), value :: odist
  integer(C_FFTW_R2R_KIND), dimension(*), intent(in) :: kind
  integer(C_INT), value :: flags
end function fftwf_plan_many_r2r
```

Returns

This function returns an `fftwf_plan` object that can be used in subsequent *fftwf_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwf_execute_r2r* `()`.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return `NULL`.

Parameters

`rank`

Input parameter

`rank` is `int`

The number of dimensions of the FFT problem to be solved

Constraint: rank >= 1

`n` Input parameter

`n` is `const int`

`n` is an array of dimension `[rank]`.

`n` contains the size of each dimension of the FFT to be solved

Constraint: `n[i]` >= 1 for i in 1, rank

`howmany` Input parameter

`howmany` is `int`

`howmany` is the number of multi-dimensional FFT transforms to be computed

Constraint: `howmany` >= 1

`in` Input parameter

`in` is `float`

`in` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`inembed` Input parameter

`inembed` is `const int`

`inembed` is a pointer to a previously allocated array of `float` data. This array denotes the dimensionality of a potentially larger space of which the input array, `in`, dimensions given by `n`, is a sub array.

Constraint: Either `inembed[i] >= n[i]` for `i` in 0, `rank-1` OR if `inembed == NULL`, then `inembed` is assumed to have the same dimensions as `n`

`istride` Input parameter

`istride` is `int`

`istride` is the distance between successive elements of the input data

`idist` Input parameter

`idist` is `int`

`idist` is the distance between the start of each of the `howmany` transforms in the input data

`out`

Input parameter

`out` is `float`

`out` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`onembed` Input parameter

`onembed` is `const int`

`onembed` is an array of dimension `[rank]` or `NULL`. This array denotes the dimensionality of a potentially larger space of which the input array, `out`, dimensions given by `n`, is a sub array.

Constraint: Either `onembed[i] >= n[i]` for `i` in 0, `rank-1` OR if `onembed == NULL`, then `onembed` is assumed to have the same dimensions as `n`

`ostride` Input parameter

`ostride` is `int`

`ostride` is the distance between successive elements of the output data

`odist` Input parameter

`odist` is `int`

`odist` is the distance between the start of each of the `howmany` transforms in the output data

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.

- FFTW_PATIENT – Produce a plan, testing more options than FFTW_MEASURE. Input and output data might be overwritten.
- FFTW_EXHAUSTIVE – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- FFTW_WISDOM_ONLY – Produce a plan only if the requested case has existing wisdom, otherwise return NULL.
- FFTW_PRESERVE_INPUT – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- FFTW_DESTROY_INPUT – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- FFTW_CONSERVE_MEMORY – Request that extra memory is not created. This might be ignored.
- FFTW_UNALIGNED – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by FFTW_MEASURE, FFTW_PATIENT and FFTW_EXHAUSTIVE will be limited when using *fftwf_set_timelimit*.
- If multiple options for planning rigor are included in flags, then the highest level of planning requested will be used.

5.6.11 fftwf_plan_r2r

fftwf_plan_r2r is a basic FFTW interface call for planning an *n*-dimensional real-to-real FFT operation on a single, contiguous data sequence.

Note: in and out can be the same pointer for an "in-place" transform.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return NULL.

Syntax

C specification:

```
#include "fftw3.h"

fftwf_plan fftwf_plan_r2r(int rank, const int *n, float *in, float *out,
                          unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_plan_r2r(rank,n,in,out,kind,flags) &
    bind(C, name='fftwf_plan_r2r')
  integer(C_INT), value :: rank
  integer(C_INT), dimension(*), intent(in) :: n
  real(C_FLOAT), dimension(*), intent(out) :: in
  real(C_FLOAT), dimension(*), intent(out) :: out
```

(continues on next page)

(continued from previous page)

```
  integer(C_FFTW_R2R_KIND), dimension(*), intent(in) :: kind
  integer(C_INT), value :: flags
end function fftwf_plan_r2r
```

Returns

This function returns an `fftwf_plan` object that can be used in subsequent *fftwf_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwf_execute_r2r* `()`.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return `NULL`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of the FFT problem to be solved

Constraint: `rank` >= 1

`n` Input parameter

`n` is `const int*`.

`n` is an array of dimension `[rank]`.

`n` contains the size of each dimension of the FFT to be solved

Constraint: `n[i]` >=1 for i in 1, `rank`

`in` Input parameter

`in` is `float`

`in` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `float`

`out` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.

- FFTW_EXHAUSTIVE – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- FFTW_WISDOM_ONLY – Produce a plan only if the requested case has existing wisdom, otherwise return NULL.
- FFTW_PRESERVE_INPUT – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- FFTW_DESTROY_INPUT – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- FFTW_CONSERVE_MEMORY – Request that extra memory is not created. This might be ignored.
- FFTW_UNALIGNED – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by FFTW_MEASURE, FFTW_PATIENT and FFTW_EXHAUSTIVE will be limited when using *fftwf_set_timelimit*.
- If multiple options for planning rigor are included in flags, then the highest level of planning requested will be used.

5.6.12 fftwf_plan_r2r_1d

fftwf_plan_r2r_1d is a basic FFTW interface call for planning a 1-d real-to-real FFT operation on a single, contiguous data sequence.

Note: in and out can be the same pointer for an "in-place" transform.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return NULL.

Syntax

C specification:

```
#include "fftw3.h"

fftwf_plan fftwf_plan_r2r_1d(int n0, float *in, float *out, unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_plan_r2r_1d(n,in,out,kind,flags) &
    bind(C, name='fftwf_plan_r2r_1d')
  integer(C_INT), value :: n
  real(C_FLOAT), dimension(*), intent(out) :: in
  real(C_FLOAT), dimension(*), intent(out) :: out
  integer(C_FFTW_R2R_KIND), value :: kind
  integer(C_INT), value :: flags
end function fftwf_plan_r2r_1d
```

Returns

This function returns an `fftwf_plan` object that can be used in subsequent *fftwf_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwf_execute_r2r* `()`.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return `NULL`.

Parameters

`n0` Input parameter

`n0` is `int`

This is the length of the FFT sequence

Constraint: `n0` >= 1

`in` Input parameter

`in` is `float`

`in` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `float`

`out` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.

- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwf_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.6.13 fftwf_plan_r2r_2d

`fftwf_plan_r2r_2d` is a basic FFTW interface call for planning a 2-d real-to-real FFT operation on a single, contiguous data sequence.

Note: `in` and `out` can be the same pointer for an "in-place" transform.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return `NULL`.

Syntax

C specification:

```
#include "fftw3.h"

fftwf_plan fftwf_plan_r2r_2d(int n0, int n1, float *in, float *out,
                             unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_plan_r2r_2d(n0,n1,in,out,kind0,kind1,flags) &
    bind(C, name='fftwf_plan_r2r_2d')
  integer(C_INT), value :: n0
  integer(C_INT), value :: n1
  real(C_FLOAT), dimension(*), intent(out) :: in
  real(C_FLOAT), dimension(*), intent(out) :: out
  integer(C_FFTW_R2R_KIND), value :: kind0
  integer(C_FFTW_R2R_KIND), value :: kind1
  integer(C_INT), value :: flags
end function fftwf_plan_r2r_2d
```

Returns

This function returns an `fftwf_plan` object that can be used in subsequent *fftwf_execute*`()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwf_execute_r2r*`()`.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return `NULL`.

Parameters

n0 Input parameter

`n0` is `int`

This is the length of the first dimension of the FFT sequence

Constraint: `n0` >= 1

n1 Input parameter

`n1` is `int`

This is the length of the second dimension of the FFT sequence

Constraint: `n1` >= 1

in Input parameter

`in` is `float`

`in` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

out Input parameter

`out` is `float`

`out` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

flags Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by FFTW_MEASURE, FFTW_PATIENT and FFTW_EXHAUSTIVE will be limited when using *fftwf_set_timelimit*.
- If multiple options for planning rigor are included in flags, then the highest level of planning requested will be used.

5.6.14 fftwf_plan_r2r_3d

fftwf_plan_r2r_3d is a basic FFTW interface call for planning a 3-d real-to-real FFT operation on a single, contiguous data sequence.

Note: in and out can be the same pointer for an "in-place" transform.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return NULL.

Syntax

C specification:

```
#include "fftw3.h"

fftwf_plan fftwf_plan_r2r_3d(int n0, int n1, int n2, float *in, float *out,
                             unsigned flags);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_plan_r2r_3d(n0,n1,n2,in,out,kind0,kind1, &
    kind2,flags) bind(C, name='fftwf_plan_r2r_3d')
  integer(C_INT), value :: n0
  integer(C_INT), value :: n1
  integer(C_INT), value :: n2
  real(C_FLOAT), dimension(*), intent(out) :: in
  real(C_FLOAT), dimension(*), intent(out) :: out
  integer(C_FFTW_R2R_KIND), value :: kind0
  integer(C_FFTW_R2R_KIND), value :: kind1
  integer(C_FFTW_R2R_KIND), value :: kind2
  integer(C_INT), value :: flags
end function fftwf_plan_r2r_3d
```

Returns

This function returns an fftwf_plan object that can be used in subsequent *fftwf_execute*() calls on the previously supplied in and out arrays, or using different arrays via *fftwf_execute_r2r*().

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return NULL.

Parameters

n0 Input parameter

`n0` is `int`

This is the length of the first dimension of the FFT sequence

Constraint: `n0` >= 1

n1 Input parameter

`n1` is `int`

This is the length of the second dimension of the FFT sequence

Constraint: `n1` >= 1

n2 Input parameter

`n2` is `int`

This is the length of the third dimension of the FFT sequence

Constraint: `n2` >= 1

in Input parameter

`in` is float

`in` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

out Input parameter

`out` is `float`

`out` is a pointer to a previously allocated array of `float` data. At `fftwf_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

flags Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.

- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwf_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.6.15 fftwh_plan_guru_r2r

`fftwh_plan_many_r2r` is a guru FFTW interface call for planning an *n*-dimensional complex-to-complex FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return `NULL`.

Syntax

C specification:

```
#include "fftw3.h"

fftwh_plan fftwh_plan_guru_r2r(int rank, const fftwh_iodim *dims,
                               int howmany_rank,
                               const fftwh_iodim *howmany_dims,
                               __fp16 *in, __fp16 *out,
                               unsigned flags);
```

Returns

This function returns an `fftwh_plan` object that can be used in subsequent *fftwh_execute*`()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwh_execute_r2r*`()`.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return `NULL`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

`dims` Input parameter

`dims` is `const fftwh_iodim *`

`dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the length of the `i` th dimension.
- `int is` – the stride between successive elements in the input vector in the `i` th dimension.
- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: `dims.n[i]` >= 1 for i in 0, `rank-1`

`howmany_rank` Input parameter

`howmany_rank` is `int`

`howmany_rank` is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then `howmany_rank` needs only be 1.

Constraint: `howmany` >= 0

Note: If `howmany_rank=0`, then a single FFT transform is computed.

`howmany_dims` Input parameter

`howmany_dims` is `const fftwh_iodim *`

`howmany_dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `int n` – the number of transforms to be computed of the `i` th dimension.of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `int is` – the number of memory elements, of type `__fp16`, between the start of transforms in the `i` th dimension. of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `int os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed is given by the product of the `howmany_dims.n[]` entries. For example, if `howmany_rank=1`, then `howmany_dims[0].n` is the memory distance between successive transforms.

Constraint: `howmany_dims.n[i]` >= 1 for i in 0, `howmany_rank-1`

`in` Input parameter

`in` is `__fp16`

`in` is a pointer to a previously allocated array of `__fp16` data. At `fftwh_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `__fp16`

`out` is a pointer to a previously allocated array of `__fp16` data. At `fftwh_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwh_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.6.16 fftwh_plan_guru64_dft

`fftwh_plan_guru64_dft` is a guru FFTW interface call for planning an *n*-dimensional real-to-real FFT operation on multiple sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner.

Unlike *fftwh_plan_guru_dft*, `fftwh_plan_guru64_dft` allows selected parameters to be specified as 64-bit integers. For Arm Performance Libraries, this parameter size difference is true in both `armpl*_lp64` and `armpl_ilp64` variants. Therefore, the parameter size difference is independent of the integer size used in the rest of the library interface.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return `NULL`.

Syntax

C specification:

```
#include "fftw3.h"

fftwh_plan fftwh_plan_guru64_dft(int rank, const fftwh_iodim64 *dims,
                                 int howmany_rank,
                                 const fftwh_iodim64 *howmany_dims,
                                 __fp16 *in, __fp16 *out,
                                 unsigned flags);
```

Returns

This function returns an `fftwh_plan` object that can be used in subsequent *fftwh_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwh_execute_dft* `()`.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return `NULL`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of each individual FFT problem to be solved.

Constraint: `rank` >= 1

`dims` Input parameter

`dims` is `const fftwh_iodim64 *`

`dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `ptrdiff_t n` – the length of the `i` th dimension.
- `ptrdiff_t is` – the stride between successive elements in the input vector in the `i` th dimension.
- `ptrdiff_t os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns of the individual, multi-dimensional FFTs to be performed.

Constraint: `dims.n[i]` >= 1 for i in 0, `rank-1`

`howmany_rank` Input parameter

`howmany_rank` is `int`

`howmany_rank` is the number of dimensions needed to describe the memory access pattern for the start addresses of each of the multi-dimensional FFT transforms to be computed. For example, if all the transforms to be performed are regularly offset in memory, then `howmany_rank` needs only be 1.

Constraint: `howmany` >= 0

Note: If `howmany_rank=0`, then a single FFT transform is computed.

`howmany_dims` Input parameter

`howmany_dims` is `const fftwh_iodim64 *`

`howmany_dims` is an array of structures, dimension `[rank]`, where the members of `dims[i]`, for `i` in 0, `rank-1`, are defined to be:

- `ptrdiff_t n` – the number of transforms to be computed of the `i` th dimension.of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `ptrdiff_t is` – the number of memory elements, of type `__fp16`, between the start of transforms in the `i` th dimension. of the `howmany_rank`-dimensional space mapped by the transforms to be computed.
- `ptrdiff_t os` – the stride between successive elements in the output vector in the `i` th dimension.

This parameters collectively define the size of each dimension and memory access patterns from the multi-dimensional array of transforms that need to be performed. The total number of transforms to be computed

is given by the product of the howmany_dims.n[] entries. For example, if howmany_rank=1, then howmany_dims[0].n is the memory distance between successive transforms.

Constraint: howmany_dims.n[i] >= 1 for i in 0, howmany_rank-1

in Input parameter

in is __fp16

in is a pointer to a previously allocated array of __fp16 data. At fftwh_execute() time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

out Input parameter

out is __fp16

out is a pointer to a previously allocated array of __fp16 data. At fftwh_execute() time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

flags Input parameter

flags is unsigned

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation.

Valid options for inclusion in flags are:

- FFTW_ESTIMATE – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- FFTW_MEASURE – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- FFTW_PATIENT – Produce a plan, testing more options than FFTW_MEASURE. Input and output data might be overwritten.
- FFTW_EXHAUSTIVE – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- FFTW_WISDOM_ONLY – Produce a plan only if the requested case has existing wisdom, otherwise return NULL.
- FFTW_PRESERVE_INPUT – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- FFTW_DESTROY_INPUT – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- FFTW_CONSERVE_MEMORY – Request that extra memory is not created. This might be ignored.
- FFTW_UNALIGNED – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by FFTW_MEASURE, FFTW_PATIENT and FFTW_EXHAUSTIVE will be limited when using *fftwh_set_timelimit*.
- If multiple options for planning rigor are included in flags, then the highest level of planning requested will be used.

5.6.17 fftwh_plan_many_r2r

`fftwh_plan_many_r2r` is an advanced FFTW interface call for planning an *n*-dimensional real-to-real FFT operation on `howmany` sets of data. The data for an individual FFT does not need to be contiguous, and can be provided in a strided manner.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return `NULL`.

Syntax

C specification:

```
#include "fftw3.h"

fftwh_plan fftwh_plan_many_r2r(int rank, const int *n, int howmany,
                               __fp16 *in, const int *inembed, int istride,
                               int idist, __fp16 *out, const int *onembed,
                               int ostride, int odist, unsigned flags);
```

Returns

This function returns an `fftwh_plan` object that can be used in subsequent *fftwh_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwh_execute_r2r* `()`.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return `NULL`.

Parameters

rank

Input parameter

`rank` is `int`

The number of dimensions of the FFT problem to be solved

Constraint: rank >= 1

n Input parameter

`n` is `const int`

`n` is an array of dimension `[rank]`.

`n` contains the size of each dimension of the FFT to be solved

Constraint: `n[i]` >= 1 for i in 1, rank

howmany Input parameter

`howmany` is `int`

`howmany` is the number of multi-dimensional FFT transforms to be computed

Constraint: `howmany` >= 1

in Input parameter

`in` is `__fp16`

`in` is a pointer to a previously allocated array of `__fp16` data. At `fftwh_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

inembed Input parameter

`inembed` is `const int`

`inembed` is a pointer to a previously allocated array of `__fp16` data. This array denotes the dimensionality of a potentially larger space of which the input array, `in`, dimensions given by `n`, is a sub array.

Constraint: Either `inembed[i] >= n[i]` for `i` in 0, `rank-1` OR if `inembed == NULL`, then `inembed` is assumed to have the same dimensions as `n`

istride Input parameter

`istride` is `int`

`istride` is the distance between successive elements of the input data

idist Input parameter

`idist` is `int`

`idist` is the distance between the start of each of the `howmany` transforms in the input data

out

Input parameter

`out` is `__fp16`

`out` is a pointer to a previously allocated array of `__fp16` data. At `fftwh_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

onembed Input parameter

`onembed` is `const int`

`onembed` is an array of dimension `[rank]` or `NULL`. This array denotes the dimensionality of a potentially larger space of which the input array, `out`, dimensions given by `n`, is a sub array.

Constraint: Either `onembed[i] >= n[i]` for `i` in 0, `rank-1` OR if `onembed == NULL`, then `onembed` is assumed to have the same dimensions as `n`

ostride Input parameter

`ostride` is `int`

`ostride` is the distance between successive elements of the output data

odist Input parameter

`odist` is `int`

`odist` is the distance between the start of each of the `howmany` transforms in the output data

flags Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.

- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwh_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.6.18 fftwh_plan_r2r

`fftwh_plan_r2r` is a basic FFTW interface call for planning an *n*-dimensional real-to-real FFT operation on a single, contiguous data sequence.

Note: `in` and `out` can be the same pointer for an "in-place" transform.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return `NULL`.

Syntax

C specification:

```
#include "fftw3.h"

fftwh_plan fftwh_plan_r2r(int rank, const int *n, __fp16 *in, __fp16 *out,
                          unsigned flags);
```

Returns

This function returns an `fftwh_plan` object that can be used in subsequent *fftwh_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwh_execute_r2r* `()`.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return `NULL`.

Parameters

`rank` Input parameter

`rank` is `int`

The number of dimensions of the FFT problem to be solved

Constraint: `rank` >= 1

`n` Input parameter

`n` is `const int*`.

`n` is an array of dimension `[rank]`.

`n` contains the size of each dimension of the FFT to be solved

Constraint: `n[i]` >=1 for i in 1, `rank`

`in` Input parameter

`in` is `__fp16`

`in` is a pointer to a previously allocated array of `__fp16` data. At `fftwh_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `__fp16`

`out` is a pointer to a previously allocated array of `__fp16` data. At `fftwh_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.

- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwh_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.6.19 fftwh_plan_r2r_1d

`fftwh_plan_r2r_1d` is a basic FFTW interface call for planning a 1-d real-to-real FFT operation on a single, contiguous data sequence.

Note: `in` and `out` can be the same pointer for an "in-place" transform.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return `NULL`.

Syntax

C specification:

```
#include "fftw3.h"

fftwh_plan fftwh_plan_r2r_1d(int n0, __fp16 *in, __fp16 *out, unsigned flags);
```

Returns

This function returns an `fftwh_plan` object that can be used in subsequent *fftwh_execute* `()` calls on the previously supplied `in` and `out` arrays, or using different arrays via *fftwh_execute_r2r* `()`.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return `NULL`.

Parameters

`n0` Input parameter

`n0` is `int`

This is the length of the FFT sequence

Constraint: `n0` >= 1

in Input parameter

`in` is `__fp16`

`in` is a pointer to a previously allocated array of `__fp16` data. At `fftwh_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

out Input parameter

`out` is `__fp16`

`out` is a pointer to a previously allocated array of `__fp16` data. At `fftwh_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

flags Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.
- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwh_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.6.20 fftwh_plan_r2r_2d

`fftwh_plan_r2r_2d` is a basic FFTW interface call for planning a 2-d real-to-real FFT operation on a single, contiguous data sequence.

Note: `in` and `out` can be the same pointer for an "in-place" transform.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return NULL.

Syntax

C specification:

```
#include "fftw3.h"

fftwh_plan fftwh_plan_r2r_2d(int n0, int n1, __fp16 *in, __fp16 *out,
                             unsigned flags);
```

Returns

This function returns an fftwh_plan object that can be used in subsequent *fftwh_execute*() calls on the previously supplied in and out arrays, or using different arrays via *fftwh_execute_r2r*().

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return NULL.

Parameters

n0 Input parameter

n0 is int

This is the length of the first dimension of the FFT sequence

Constraint: n0 >= 1

n1 Input parameter

n1 is int

This is the length of the second dimension of the FFT sequence

Constraint: n1 >= 1

in Input parameter

in is __fp16

in is a pointer to a previously allocated array of __fp16 data. At fftwh_execute() time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

out Input parameter

out is __fp16

out is a pointer to a previously allocated array of __fp16 data. At fftwh_execute() time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

flags Input parameter

flags is unsigned

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in flags are:

- FFTW_ESTIMATE – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- FFTW_MEASURE – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- FFTW_PATIENT – Produce a plan, testing more options than FFTW_MEASURE. Input and output data might be overwritten.
- FFTW_EXHAUSTIVE – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- FFTW_WISDOM_ONLY – Produce a plan only if the requested case has existing wisdom, otherwise return NULL.
- FFTW_PRESERVE_INPUT – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- FFTW_DESTROY_INPUT – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- FFTW_CONSERVE_MEMORY – Request that extra memory is not created. This might be ignored.
- FFTW_UNALIGNED – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by FFTW_MEASURE, FFTW_PATIENT and FFTW_EXHAUSTIVE will be limited when using *fftwh_set_timelimit*.
- If multiple options for planning rigor are included in flags, then the highest level of planning requested will be used.

5.6.21 fftwh_plan_r2r_3d

fftwh_plan_r2r_3d is a basic FFTW interface call for planning a 3-d real-to-real FFT operation on a single, contiguous data sequence.

Note: in and out can be the same pointer for an "in-place" transform.

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return NULL.

Syntax

C specification:

```
#include "fftw3.h"

fftwh_plan fftwh_plan_r2r_3d(int n0, int n1, int n2, __fp16 *in, __fp16 *out,
                             unsigned flags);
```

Returns

This function returns an fftwh_plan object that can be used in subsequent *fftwh_execute*() calls on the previously supplied in and out arrays, or using different arrays via *fftwh_execute_r2r*().

Note: Real-to-real transforms are not available in Arm Performance Libraries at present and will always return `NULL`.

Parameters

`n0` Input parameter

`n0` is `int`

This is the length of the first dimension of the FFT sequence

Constraint: `n0` >= 1

`n1` Input parameter

`n1` is `int`

This is the length of the second dimension of the FFT sequence

Constraint: `n1` >= 1

`n2` Input parameter

`n2` is `int`

This is the length of the third dimension of the FFT sequence

Constraint: `n2` >= 1

`in` Input parameter

`in` is __fp16

`in` is a pointer to a previously allocated array of `__fp16` data. At `fftwh_execute()` time this will have been filled with the input sequence on which to perform the FFT.

Storage: Data in arrays will be stored in row-major order.

`out` Input parameter

`out` is `__fp16`

`out` is a pointer to a previously allocated array of `__fp16` data. At `fftwh_execute()` time this will be filled with the resulting transform coefficients

Storage: Data in arrays will be stored in row-major order.

`flags` Input parameter

`flags` is `unsigned`

This variable combines various options that the user can supply to aid the planning. These can be combined using a bitwise OR operation. Valid options for inclusion in `flags` are:

- `FFTW_ESTIMATE` – Produce a plan quickly, without testing how good the chosen option is. Input and output data is guaranteed to be preserved.
- `FFTW_MEASURE` – Produce a plan, having tested a small number of options to find the fastest choice. Input and output data might be overwritten.
- `FFTW_PATIENT` – Produce a plan, testing more options than `FFTW_MEASURE`. Input and output data might be overwritten.
- `FFTW_EXHAUSTIVE` – Produce a plan, having tested as wide a variety of cases as possible, in order to find the best performing method.
- `FFTW_WISDOM_ONLY` – Produce a plan only if the requested case has existing wisdom, otherwise return `NULL`.

- `FFTW_PRESERVE_INPUT` – On execution of an out-of-place transform, insist that input data is not overwritten. For Arm Performance Libraries this the default in all cases.
- `FFTW_DESTROY_INPUT` – Allow planning to overwrite input data in executing out-of-place transforms. This is not needed in Arm Performance Libraries because input data is not overwritten for these transforms.
- `FFTW_CONSERVE_MEMORY` – Request that extra memory is not created. This might be ignored.
- `FFTW_UNALIGNED` – Indicate that arrays might be aligned differently at execution time than at planning, and that plans should account for this.

Notes:

- The planning times indicated by `FFTW_MEASURE`, `FFTW_PATIENT` and `FFTW_EXHAUSTIVE` will be limited when using *fftwh_set_timelimit*.
- If multiple options for planning rigor are included in `flags`, then the highest level of planning requested will be used.

5.7 FFT executors functions

5.7.1 fftw_execute

`fftw_execute` executes a previously created FFT plan.

Syntax

C specification:

```
#include "fftw3.h"

void fftw_execute(const fftw_plan p);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

 subroutine fftw_execute_dft(p,in,out) bind(C, name='fftw_execute_dft')
   type(C_PTR), value :: p
   complex(C_DOUBLE_COMPLEX), dimension(*), intent(inout) :: in
   complex(C_DOUBLE_COMPLEX), dimension(*), intent(out) :: out
 end subroutine fftw_execute_dft
```

Parameters

p Input parameter

`p` is `fftw_plan`

An FFT plan created from a preceding call

5.7.2 fftw_execute_dft

`fftw_execute_dft` executes a previously created FFT plan, however input and output data locations maybe different than those given at planning time.

Syntax

C specification:

```
#include "fftw3.h"

void fftw_execute_dft(const fftw_plan p, fftw_complex *in,
                      fftw_complex *out);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

subroutine fftw_execute_dft(p,in,out) bind(C, name='fftw_execute_dft')
  type(C_PTR), value :: p
  complex(C_DOUBLE_COMPLEX), dimension(*), intent(inout) :: in
  complex(C_DOUBLE_COMPLEX), dimension(*), intent(out) :: out
end subroutine fftw_execute_dft
```

Parameters

p Input parameter

p is fftw_plan

An FFT plan created from a preceding call

in Input parameter

in is fftw_complex*

in is a pointer to a previously allocated array of data, on which to calculate a complex-to-complex FFT

out Output parameter

out is fftw_complex*

out is a pointer to a previously allocated array of data, into which to return the results of the complex-to-complex FFT

5.7.3 fftw_execute_dft_c2r

fftw_execute_dft_c2r executes a previously created complex-to-real FFT plan, however input and output data locations maybe different than those given at planning time.

Syntax

C specification:

```
#include "fftw3.h"

void fftw_execute_dft_c2r(const fftw_plan p, fftw_complex *in,
                          double *out);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

subroutine fftw_execute_dft_c2r(p,in,out) &
```

(continues on next page)

(continued from previous page)

```
    bind(C, name='fftw_execute_dft_c2r')
  type(C_PTR), value :: p
  complex(C_DOUBLE_COMPLEX), dimension(*), intent(inout) :: in
  real(C_DOUBLE), dimension(*), intent(out) :: out
end subroutine fftw_execute_dft_c2r
```

Parameters

p Input parameter

p is fftw_plan

An FFT plan created from a preceding call

in Input parameter

in is fftw_complex*

in is a pointer to a previously allocated array of data, on which to calculate a complex-to-real FFT

out output out is double*

out is a pointer to a previously allocated array of data, into which to return the results of the complex-to-real FFT

5.7.4 fftw_execute_dft_r2c

fftw_execute_dft_r2c executes a previously created FFT plan, however input and output data locations maybe different than those given at planning time.

Syntax

C specification:

```
#include "fftw3.h"

void fftw_execute_dft_r2c(const fftw_plan p, double *in,
                          fftw_complex *out);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

subroutine fftw_execute_dft_r2c(p,in,out) &
    bind(C, name='fftw_execute_dft_r2c')
  type(C_PTR), value :: p
  real(C_DOUBLE), dimension(*), intent(inout) :: in
  complex(C_DOUBLE_COMPLEX), dimension(*), intent(out) :: out
end subroutine fftw_execute_dft_r2c
```

Parameters

p Input parameter

p is fftw_plan

An FFT plan created from a preceding call

in Input parameter

`in` is `double*`

`in` is a pointer to a previously allocated array of data, on which to calculate a real-to-complex FFT

out Output parameter

`out` is `fftw_complex*`

`out` is a pointer to a previously allocated array of data, into which to return the results of the real-to-complex FFT

5.7.5 fftw_execute_r2r

`fftw_execute_r2r` executes a previously created real-to-real FFT plan, however input and output data locations maybe different than those given at planning time.

Note: Since real-to-real transforms are not currently available in Arm Performance Libraries it is impossible to provide a valid plan, `p`, to pass to this function.

Syntax

C specification:

```
#include "fftw3.h"

void fftw_execute_r2r(const fftw_plan p, double *in, double *out);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

subroutine fftw_execute_r2r(p,in,out) bind(C, name='fftw_execute_r2r')
  type(C_PTR), value :: p
  real(C_DOUBLE), dimension(*), intent(inout) :: in
  real(C_DOUBLE), dimension(*), intent(out) :: out
end subroutine fftw_execute_r2r
```

Parameters

p Input parameter

`p` is `fftw_plan`

An FFT plan created from a preceding call

in Input parameter

`in` is `double*`

`in` is a pointer to a previously allocated array of data, on which to calculate a real-to-real FFT

out Output parameter

`out` is `double*`

`out` is a pointer to a previously allocated array of data, into which to return the results of the real-to-real FFT

5.7.6 fftwf_execute

`fftwf_execute` executes a previously created FFT plan.

Syntax

C specification:

```
#include "fftw3.h"

void fftwf_execute(const fftwf_plan p);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

subroutine fftwf_execute_dft(p,in,out) bind(C, name='fftwf_execute_dft')
  type(C_PTR), value :: p
  complex(C_FLOAT_COMPLEX), dimension(*), intent(inout) :: in
  complex(C_FLOAT_COMPLEX), dimension(*), intent(out) :: out
end subroutine fftwf_execute_dft
```

Parameters

p Input parameter

`p` is `fftwf_plan`

An FFT plan created from a preceding call

5.7.7 fftwf_execute_dft

`fftwf_execute_dft` executes a previously created FFT plan, however input and output data locations maybe different than those given at planning time.

Syntax

C specification:

```
#include "fftw3.h"

void fftwf_execute_dft(const fftwf_plan p, fftwf_complex *in,
                       fftwf_complex *out);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

subroutine fftwf_execute_dft(p,in,out) bind(C, name='fftwf_execute_dft')
  type(C_PTR), value :: p
  complex(C_FLOAT_COMPLEX), dimension(*), intent(inout) :: in
  complex(C_FLOAT_COMPLEX), dimension(*), intent(out) :: out
end subroutine fftwf_execute_dft
```

Parameters

p Input parameter

p is fftwf_plan

An FFT plan created from a preceding call

in Input parameter

in is fftwf_complex*

in is a pointer to a previously allocated array of data, on which to calculate a complex-to-complex FFT

out Output parameter

out is fftwf_complex*

out is a pointer to a previously allocated array of data, into which to return the results of the complex-to-complex FFT

5.7.8 fftwf_execute_dft_c2r

fftwf_execute_dft_c2r executes a previously created complex-to-real FFT plan, however input and output data locations maybe different than those given at planning time.

Syntax

C specification:

```
#include "fftw3.h"

void fftwf_execute_dft_c2r(const fftwf_plan p, fftwf_complex *in,
                           float *out);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

subroutine fftwf_execute_dft_c2r(p,in,out) bind(C, name='fftwf_execute_dft_c2r')
  type(C_PTR), value :: p
  complex(C_FLOAT_COMPLEX), dimension(*), intent(inout) :: in
  real(C_FLOAT), dimension(*), intent(out) :: out
end subroutine fftwf_execute_dft_c2r
```

Parameters

p Input parameter

p is fftwf_plan

An FFT plan created from a preceding call

in Input parameter

in is fftwf_complex*

in is a pointer to a previously allocated array of data, on which to calculate a complex-to-real FFT

out output out is float*

out is a pointer to a previously allocated array of data, into which to return the results of the complex-to-real FFT

5.7.9 fftwf_execute_dft_r2c

`fftwf_execute_dft_r2c` executes a previously created FFT plan, however input and output data locations maybe different than those given at planning time.

Syntax

C specification:

```
#include "fftw3.h"

void fftwf_execute_dft_r2c(const fftwf_plan p, float *in,
                           fftwf_complex *out);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

subroutine fftwf_execute_dft_r2c(p,in,out) &
    bind(C, name='fftwf_execute_dft_r2c')
  type(C_PTR), value :: p
  real(C_FLOAT), dimension(*), intent(inout) :: in
  complex(C_FLOAT_COMPLEX), dimension(*), intent(out) :: out
end subroutine fftwf_execute_dft_r2c
```

Parameters

p Input parameter

`p` is `fftwf_plan`

An FFT plan created from a preceding call

in Input parameter

`in` is `float*`

`in` is a pointer to a previously allocated array of data, on which to calculate a real-to-complex FFT

out Output parameter

`out` is `fftwf_complex*`

`out` is a pointer to a previously allocated array of data, into which to return the results of the real-to-complex FFT

5.7.10 fftwf_execute_r2r

`fftwf_execute_r2r` executes a previously created real-to-real FFT plan, however input and output data locations maybe different than those given at planning time.

Note: Since real-to-real transforms are not currently available in Arm Performance Libraries it is impossible to provide a valid plan, `p`, to pass to this function.

Syntax

C specification:

```
#include "fftw3.h"

void fftwf_execute_r2r(const fftwf_plan p, float *in, float *out);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

subroutine fftwf_execute_r2r(p,in,out) bind(C, name='fftwf_execute_r2r')
  type(C_PTR), value :: p
  real(C_FLOAT), dimension(*), intent(inout) :: in
  real(C_FLOAT), dimension(*), intent(out) :: out
end subroutine fftwf_execute_r2r
```

Parameters

p Input parameter

p is fftwf_plan

An FFT plan created from a preceding call

in Input parameter

in is float*

in is a pointer to a previously allocated array of data, on which to calculate a real-to-real FFT

out Output parameter

out is float*

out is a pointer to a previously allocated array of data, into which to return the results of the real-to-real FFT

5.7.11 fftwh_execute

fftwh_execute executes a previously created FFT plan.

Syntax

C specification:

```
#include "fftw3.h"

void fftwh_execute(const fftwh_plan p);
```

Parameters

p Input parameter

p is fftwh_plan

An FFT plan created from a preceding call

5.7.12 fftwh_execute_dft

`fftwh_execute_dft` executes a previously created FFT plan, however input and output data locations maybe different than those given at planning time.

Syntax

C specification:

```
#include "fftw3.h"

void fftwh_execute_dft(const fftwh_plan p, fftwh_complex *in,
                       fftwh_complex *out);
```

Parameters

p Input parameter

`p` is `fftwh_plan`

An FFT plan created from a preceding call

in Input parameter

`in` is `fftwh_complex*`

`in` is a pointer to a previously allocated array of data, on which to calculate a complex-to-complex FFT

out Output parameter

`out` is `fftwh_complex*`

`out` is a pointer to a previously allocated array of data, into which to return the results of the complex-to-complex FFT

5.7.13 fftwh_execute_dft_c2r

`fftwh_execute_dft_c2r` executes a previously created complex-to-real FFT plan, however input and output data locations maybe different than those given at planning time.

Syntax

C specification:

```
#include "fftw3.h"

void fftwh_execute_dft_c2r(const fftwh_plan p, fftwh_complex *in,
                           __fp16 *out);
```

Parameters

p Input parameter

`p` is `fftwh_plan`

An FFT plan created from a preceding call

in Input parameter

`in` is `fftwh_complex*`

`in` is a pointer to a previously allocated array of data, on which to calculate a complex-to-real FFT

out output `out` is `__fp16*`

`out` is a pointer to a previously allocated array of data, into which to return the results of the complex-to-real FFT

5.7.14 fftwh_execute_dft_r2c

`fftwh_execute_dft_r2c` executes a previously created FFT plan, however input and output data locations maybe different than those given at planning time.

Syntax

C specification:

```
#include "fftw3.h"

void fftwh_execute_dft_r2c(const fftwh_plan p, __fp16 *in,
                           fftwh_complex *out);
```

Parameters

p Input parameter

`p` is `fftwh_plan`

An FFT plan created from a preceding call

in Input parameter

`in` is `__fp16*`

`in` is a pointer to a previously allocated array of data, on which to calculate a real-to-complex FFT

out Output parameter

`out` is `fftwh_complex*`

`out` is a pointer to a previously allocated array of data, into which to return the results of the real-to-complex FFT

5.7.15 fftwh_execute_r2r

`fftwh_execute_r2r` executes a previously created real-to-real FFT plan, however input and output data locations maybe different than those given at planning time.

Note: Since real-to-real transforms are not currently available in Arm Performance Libraries it is impossible to provide a valid plan, `p`, to pass to this function.

Syntax

C specification:

```
#include "fftw3.h"

void fftwh_execute_r2r(const fftwh_plan p, __fp16 *in, __fp16 *out);
```

Parameters

p Input parameter

`p` is `fftwh_plan`

An FFT plan created from a preceding call

in Input parameter

`in` is `__fp16*`

`in` is a pointer to a previously allocated array of data, on which to calculate a real-to-real FFT

out Output parameter

`out` is `__fp16*`

`out` is a pointer to a previously allocated array of data, into which to return the results of the real-to-real FFT

5.8 FFT memory functions

5.8.1 fftw_alignment_of

`fftw_alignment_of` gives the alignment of a supplied pointer. Multiple invocations with different pointers can be used to check if arrays are aligned similarly.

Syntax

C specification:

```
#include "fftw3.h"

int fftw_alignment_of(double *p);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

integer(C_INT) function fftw_alignment_of(p) &
    bind(C, name='fftw_alignment_of')
  real(C_DOUBLE), dimension(*), intent(out) :: p
end function fftw_alignment_of
```

Returns

`fftw_alignment_of` returns an `int` denoting the alignment of the supplied pointer.

Parameters

p Input parameter

p is double

This is a pointer to a previously allocated array.

5.8.2 fftw_alloc_complex

fftw_alloc_complex allocates a block of data of the requested size.

Syntax

C specification:

```
#include "fftw3.h"

fftw_complex* fftw_alloc_complex(size_t n);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_alloc_complex(n) &
    bind(C, name='fftw_alloc_complex')
  integer(C_SIZE_T), value :: n
end function fftw_alloc_complex
```

Returns

This function returns a pointer, of type fftw_complex*, to an allocated block of memory of the requested size.

Parameters

n Input parameter

n is size_t

This is the size, in terms of number of fftw_complex entries, of array to be created

Constraint: n >= 1

5.8.3 fftw_alloc_real

fftw_alloc_real allocates a block of data of the requested size.

Syntax

C specification:

```
#include "fftw3.h"

double* fftw_alloc_real(size_t n);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_alloc_real(n) bind(C, name='fftw_alloc_real')
  integer(C_SIZE_T), value :: n
end function fftw_alloc_real
```

Returns

This function returns a pointer, of type `double*`, to an allocated block of memory of the requested size.

Parameters

n Input parameter

`n` is `size_t`

This is the size, in terms of number of `double` entries, of array to be created

Constraint: `n` >= 1

5.8.4 fftw_cleanup

`fftw_cleanup` is allowed to invalidate all FFT plans that have been previously created, without freeing memory. We recommend *not* to use this function, and in the current Arm Performance Libraries implementation this function does nothing, returning immediately.

Syntax

C specification:

```
#include "fftw3.h"

void fftw_cleanup(void);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

subroutine fftw_cleanup() bind(C, name='fftw_cleanup')
end subroutine fftw_cleanup
```

5.8.5 fftw_destroy_plan

`fftw_destroy_plan` frees all memory associated with the creation of an FFT plan. The plan should not be reused after this call.

Syntax

C specification:

```
#include "fftw3.h"

void fftw_destroy_plan(fftw_plan p);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

subroutine fftw_destroy_plan(p) bind(C, name='fftw_destroy_plan')
  type(C_PTR), value :: p
end subroutine fftw_destroy_plan
```

Parameters

p Input parameter

p is `fftw_plan`

p is a pointer to a previously created plan.

5.8.6 fftw_free

`fftw_free` frees the memory created in a previous malloc() or fftw_malloc() call.

Syntax

C specification:

```
#include "fftw3.h"

void* fftw_free(void *p);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

subroutine fftw_free(p) bind(C, name='fftw_free')
  type(C_PTR), value :: p
end subroutine fftw_free
```

Parameters

p Input parameter

p is `void *`

p is a pointer to a previously created array

5.8.7 fftw_malloc

`fftw_malloc` allocates a block of data of the requested size.

Syntax

C specification:

```
#include "fftw3.h"

void* fftw_malloc(size_t n);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_malloc(n) bind(C, name='fftw_malloc')
  integer(C_SIZE_T), value :: n
end function fftw_malloc
```

Returns

This function returns a pointer, of type `void*`, to an allocated block of memory of the requested size.

Parameters

n Input parameter

`n` is `size_t`

This is the size, in bytes, of array to be created

Constraint: `n0` >= 1

5.8.8 fftwf_alignment_of

`fftwf_alignment_of` gives the alignment of a supplied pointer. Multiple invocations with different pointers can be used to check if arrays are aligned similarly.

Syntax

C specification:

```
#include "fftw3.h"

int fftwf_alignment_of(float *p);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

integer(C_INT) function fftwf_alignment_of(p) &
    bind(C, name='fftwf_alignment_of')
  real(C_FLOAT), dimension(*), intent(out) :: p
end function fftwf_alignment_of
```

Returns

`fftwf_alignment_of` returns an `int` denoting the alignment of the supplied pointer.

Parameters

p Input parameter

`p` is `float`

This is a pointer to a previously allocated array.

5.8.9 fftwf_alloc_complex

`fftwf_alloc_complex` allocates a block of data of the requested size.

Syntax

C specification:

```
#include "fftw3.h"

fftwf_complex* fftwf_alloc_complex(size_t n);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_alloc_complex(n) &
    bind(C, name='fftwf_alloc_complex')
  integer(C_SIZE_T), value :: n
end function fftwf_alloc_complex
```

Returns

This function returns a pointer, of type `fftwf_complex*`, to an allocated block of memory of the requested size.

Parameters

n Input parameter

`n` is `size_t`

This is the size, in terms of number of `fftwf_complex` entries, of array to be created

Constraint: `n` >= 1

5.8.10 fftwf_alloc_real

`fftwf_alloc_real` allocates a block of data of the requested size.

Syntax

C specification:

```
#include "fftw3.h"

float* fftwf_alloc_real(size_t n);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_alloc_real(n) bind(C, name='fftwf_alloc_real')
  integer(C_SIZE_T), value :: n
end function fftwf_alloc_real
```

Returns

This function returns a pointer, of type `float*`, to an allocated block of memory of the requested size.

Parameters

n Input parameter

`n` is `size_t`

This is the size, in terms of number of `float` entries, of array to be created

Constraint: `n` >= 1

5.8.11 fftwf_cleanup

`fftwf_cleanup` is allowed to invalidate all FFT plans that have been previously created, without freeing memory. We recommend *not* to use this function, and in the current Arm Performance Libraries implementation this function does nothing, returning immediately.

Syntax

C specification:

```
#include "fftw3.h"

void fftwf_cleanup(void);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

subroutine fftwf_cleanup() bind(C, name='fftwf_cleanup')
end subroutine fftwf_cleanup
```

5.8.12 fftwf_destroy_plan

`fftwf_destroy_plan` frees all memory associated with the creation of an FFT plan. The plan should not be reused after this call.

Syntax

C specification:

```
#include "fftw3.h"

void fftwf_destroy_plan(fftwf_plan p);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

subroutine fftwf_destroy_plan(p) bind(C, name='fftwf_destroy_plan')
  type(C_PTR), value :: p
end subroutine fftwf_destroy_plan
```

Parameters

p Input parameter

p is fftwf_plan

p is a pointer to a previously created plan.

5.8.13 fftwf_free

fftwf_free frees the memory created in a previous malloc() or fftwf_malloc() call.

Syntax

C specification:

```
#include "fftw3.h"

void* fftwf_free(void *p);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

subroutine fftwf_free(p) bind(C, name='fftwf_free')
  type(C_PTR), value :: p
end subroutine fftwf_free
```

Parameters

p Input parameter

p is void *

p is a pointer to a previously created array

5.8.14 fftwf_malloc

fftwf_malloc allocates a block of data of the requested size.

Syntax

C specification:

```
#include "fftw3.h"

void* fftwf_malloc(size_t n);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_malloc(n) bind(C, name='fftwf_malloc')
  integer(C_SIZE_T), value :: n
end function fftwf_malloc
```

Returns

This function returns a pointer, of type `void*`, to an allocated block of memory of the requested size.

Parameters

n Input parameter

`n` is `size_t`

This is the size, in bytes, of array to be created

Constraint: `n0` >= 1

5.8.15 fftwh_alignment_of

`fftwh_alignment_of` gives the alignment of a supplied pointer. Multiple invocations with different pointers can be used to check if arrays are aligned similarly.

Syntax

C specification:

```
#include "fftw3.h"

int fftwh_alignment_of(__fp16 *p);
```

Returns

`fftwh_alignment_of` returns an `int` denoting the alignment of the supplied pointer.

Parameters

p Input parameter

`p` is `__fp16`

This is a pointer to a previously allocated array.

5.8.16 fftwh_alloc_complex

`fftwh_alloc_complex` allocates a block of data of the requested size.

Syntax

C specification:

```
#include "fftw3.h"

fftwh_complex* fftwh_alloc_complex(size_t n);
```

Returns

This function returns a pointer, of type `fftwh_complex*`, to an allocated block of memory of the requested size.

Parameters

n Input parameter

n is size_t

This is the size, in terms of number of fftwh_complex entries, of array to be created

Constraint: n >= 1

5.8.17 fftwh_alloc_real

fftwh_alloc_real allocates a block of data of the requested size.

Syntax

C specification:

```
#include "fftw3.h"

__fp16* fftwh_alloc_real(size_t n);
```

Returns

This function returns a pointer, of type __fp16*, to an allocated block of memory of the requested size.

Parameters

n Input parameter

n is size_t

This is the size, in terms of number of __fp16 entries, of array to be created

Constraint: n >= 1

5.8.18 fftwh_cleanup

fftwh_cleanup is allowed to invalidate all FFT plans that have been previously created, without freeing memory. We recommend *not* to use this function, and in the current Arm Performance Libraries implementation this function does nothing, returning immediately.

Syntax

C specification:

```
#include "fftw3.h"

void fftwh_cleanup(void);
```

5.8.19 fftwh_destroy_plan

fftwh_destroy_plan frees all memory associated with the creation of an FFT plan. The plan should not be reused after this call.

Syntax

C specification:

```
#include "fftw3.h"

void fftwh_destroy_plan(fftwh_plan p);
```

Parameters

p Input parameter

p is `fftwh_plan`

p is a pointer to a previously created plan.

5.8.20 fftwh_free

`fftwh_free` frees the memory created in a previous malloc() or fftwh_malloc() call.

Syntax

C specification:

```
#include "fftw3.h"

void* fftwh_free(void *p);
```

Parameters

p Input parameter

p is `void *`

p is a pointer to a previously created array

5.8.21 fftwh_malloc

`fftwh_malloc` allocates a block of data of the requested size.

Syntax

C specification:

```
#include "fftw3.h"

void* fftwh_malloc(size_t n);
```

Returns

This function returns a pointer, of type `void*`, to an allocated block of memory of the requested size.

Parameters

n Input parameter

n is size_t

This is the size, in bytes, of array to be created

Constraint: n0 >= 1

5.9 FFT wisdom functions

5.9.1 fftw_export_wisdom

fftw_export_wisdom calls the user-supplied function to write Arm Performance Libraries FFT wisdom out, one character at a time.

Syntax

C specification:

```
#include "fftw3.h"

void fftw_export_wisdom(void (*write_char)(char c, void *), void *data);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

subroutine fftw_export_wisdom(write_char,data) &
    bind(C, name='fftw_export_wisdom')
  type(C_FUNPTR), value :: write_char
  type(C_PTR), value :: data
end subroutine fftw_export_wisdom
```

Parameters

write_char Input parameter

write_char is a pointer to a function that will write out the wisdom accumulated by the program, one character at a time

Constraint: write_char takes two parameters:

the first is a char which is the character to be written

the second is a pointer to where this data is to be written. Examples of using this would be a FILE*, an array or unused and just writing to the screen

data Input parameter

data is a void * pointer that is itself passed as the second parameter to the function write_char

5.9.2 fftw_export_wisdom_to_file

fftw_export_wisdom_to_file exports the currently known Arm Performance Libraries FFT wisdom to the file opened as output_file specified by the user. Note that this routine expects a file pointer to an already open file, rather than *_armpl_fftw_export_wisdom_to_filname* which opens a new file with the supplied name.

Syntax

C specification:

```
#include "fftw3.h"

void fftw_export_wisdom_to_file(FILE *output_file);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

subroutine fftw_export_wisdom_to_file(output_file) &
    bind(C, name='fftw_export_wisdom_to_file')
  type(C_PTR), value :: output_file
end subroutine fftw_export_wisdom_to_file
```

Parameters

output_file Input parameter

`output_file` is `FILE *`

`output_file` is a pointer to a previously opened file.

Constraint: The file pointer `output_file` must be opened previous to calling `fftw_export_wisdom_to_file`.

5.9.3 _armpl_fftw_export_wisdom_to_filname

`fftw_export_wisdom_to_filename` exports the currently known Arm Performance Libraries FFT wisdom to a new file called `output_file` specified by the user. Note that this routine opens a new file with the supplied name, rather than *fftw_export_wisdom_to_file* which expects a file pointer to an already open file.

Syntax

C specification:

```
#include "fftw3.h"

int fftw_export_wisdom_to_filename(const char *filename);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

integer(C_INT) function fftw_export_wisdom_to_filename(filename) &
    bind(C, name='fftw_export_wisdom_to_filename')
  character(C_CHAR), dimension(*), intent(in) :: filename
end function fftw_export_wisdom_to_filename
```

Returns

If the wisdom file is successfully created, the function returns `1`. Otherwise, the function returns `0`.

Parameters

`filename` Input parameter

`filename` is `const char *`

`filename` is a pointer to an array storing the name of the file to be opened

5.9.4 fftw_export_wisdom_to_string

`fftw_export_wisdom_to_string` exports the currently known Arm Performance Libraries FFT wisdom to a character string to be returned to the user.

Syntax

C specification:

```
#include "fftw3.h"

char *fftw_export_wisdom_to_string(void);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftw_export_wisdom_to_string() &
    bind(C, name='fftw_export_wisdom_to_string')
end function fftw_export_wisdom_to_string
```

Returns

`fftw_export_wisdom_to_string` returns a pointer to a string containing the previously collected wisdom.

5.9.5 fftw_forget_wisdom

`fftw_forget_wisdom` is intended to allow a user to discard previously loaded or learned wisdom. At present, in the Arm Performance Libraries implementation no action is taken, and the function returns no error.

Syntax

C specification:

```
#include "fftw3.h"

void fftw_forget_wisdom(void);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

subroutine fftw_forget_wisdom() bind(C, name='fftw_forget_wisdom')
end subroutine fftw_forget_wisdom
```

Returns

`fftw_forget_wisdom` always returns 0.

5.9.6 fftw_import_system_wisdom

`fftw_import_system_wisdom` reads previously written Arm Performance Libraries FFT wisdom from a default system file. At present, system wisdom is unavailable for Arm Performance Libraries and this function will return 0.

Syntax

C specification:

```
#include "fftw3.h"

int fftw_import_system_wisdom(void);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

integer(C_INT) function fftw_import_system_wisdom() &
    bind(C, name='fftw_import_system_wisdom')
end function fftw_import_system_wisdom
```

Returns

`fftw_import_system_wisdom` returns either 0 indicating that the system wisdom file was read correctly, or 1 otherwise. However since system wisdom is currently unavailable for Arm Performance Libraries, this function will return 0.

5.9.7 fftw_import_wisdom

`fftw_import_wisdom` calls the user-supplied function to read Arm Performance Libraries FFT wisdom in, one character at a time.

Syntax

C specification:

```
#include "fftw3.h"

int fftw_import_wisdom(int (*read_char)(void *), void *data);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

integer(C_INT) function fftw_import_wisdom(read_char,data) &
    bind(C, name='fftw_import_wisdom')
  type(C_FUNPTR), value :: read_char
  type(C_PTR), value :: data
end function fftw_import_wisdom
```

Returns

`fftw_import_wisdom` returns either 0 indicating that the wisdom was read correctly, or 1 otherwise.

Parameters

`read_char` Input parameter

`read_char` is a pointer to a function that will read in the wisdom, one character at a time

Constraint: `write_char` takes two parameters:

the first is a `char` which is the character to be read

the second is a pointer to where this data should be read. Examples of using this would be a `FILE*` or an array

`data` Input parameter

`data` is a `void * pointer that is itself passed as the second parameter to the function ``read_char`

5.9.8 fftw_import_wisdom_from_file

`fftw_import_wisdom_from_file` reads the previously written Arm Performance Libraries FFT wisdom from the file open as `input_file`.

Syntax

C specification:

```
#include "fftw3.h"

int fftw_import_wisdom_from_file(FILE *input_file);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

integer(C_INT) function fftw_import_wisdom_from_file(input_file) &
    bind(C, name='fftw_import_wisdom_from_file')
  type(C_PTR), value :: input_file
end function fftw_import_wisdom_from_file
```

Returns

`fftw_import_wisdom_from_file` returns either 0 indicating that the wisdom file was read correctly, or 1 otherwise

Parameters

`input_file` Input parameter

`input_file` is `FILE *`

`input_file` is a pointer to a previously created a file of Arm Performance Libraries FFT wisdom

5.9.9 fftw_import_wisdom_from_filename

`fftw_import_wisdom_from_filename` reads the previously written Arm Performance Libraries FFT wisdom from a file called `input_file`.

Syntax

C specification:

```
#include "fftw3.h"

int fftw_import_wisdom_from_filename(const char *filename);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

integer(C_INT) function fftw_import_wisdom_from_filename(filename) &
    bind(C, name='fftw_import_wisdom_from_filename')
  character(C_CHAR), dimension(*), intent(in) :: filename
end function fftw_import_wisdom_from_filename
```

Returns

`fftw_import_wisdom_from_filename` returns either 0 indicating that the wisdom file was read correctly, or 1 otherwise

Parameters

`filename` Input parameter

`filename` is `const char *`

`filename` provides the name of the file from which wisdom will be loaded

5.9.10 fftw_import_wisdom_from_string

`fftw_import_wisdom_from_string` reads the previously written Arm Performance Libraries FFT wisdom from a string given by `input_string`.

Syntax

C specification:

```
#include "fftw3.h"

int fftw_import_wisdom_from_string(const char *input_string);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

integer(C_INT) function fftw_import_wisdom_from_string(input_string) &
    bind(C, name='fftw_import_wisdom_from_string')
```

(continues on next page)

(continued from previous page)

```
  character(C_CHAR), dimension(*), intent(in) :: input_string
end function fftw_import_wisdom_from_string
```

Returns

`fftw_import_wisdom_from_string` returns either 0 indicating that the wisdom file was read correctly, or 1 otherwise

Parameters

`input_string` Input parameter

`input_string` is `const char *`

`input_string` is a pointer to a previously created string from which the accumulated wisdom will be read

5.9.11 fftwf_export_wisdom

`fftwf_export_wisdom` calls the user-supplied function to write Arm Performance Libraries FFT wisdom out, one character at a time.

Syntax

C specification:

```
#include "fftw3.h"

void fftwf_export_wisdom(void (*write_char)(char c, void *), void *data);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

subroutine fftwf_export_wisdom(write_char,data) &
    bind(C, name='fftwf_export_wisdom')
  type(C_FUNPTR), value :: write_char
  type(C_PTR), value :: data
end subroutine fftwf_export_wisdom
```

Parameters

`write_char` Input parameter

`write_char` is a pointer to a function that will write out the wisdom accumulated by the program, one character at a time

Constraint: `write_char` takes two parameters:

the first is a `char` which is the character to be written

the second is a pointer to where this data is to be written. Examples of using this would be a `FILE*`, an array or unused and just writing to the screen

`data` Input parameter

`data` is a `void *` pointer that is itself passed as the second parameter to the function `write_char`

5.9.12 fftwf_export_wisdom_to_file

`fftwf_export_wisdom_to_file` exports the currently known Arm Performance Libraries FFT wisdom to the file opened as `output_file` specified by the user. Note that this routine expects a file pointer to an already open file, rather than *_armpl_fftwf_export_wisdom_to_filname* which opens a new file with the supplied name.

Syntax

C specification:

```
#include "fftw3.h"

void fftwf_export_wisdom_to_file(FILE *output_file);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

subroutine fftwf_export_wisdom_to_file(output_file) &
    bind(C, name='fftwf_export_wisdom_to_file')
  type(C_PTR), value :: output_file
end subroutine fftwf_export_wisdom_to_file
```

Parameters

`output_file` Input parameter

`output_file` is `FILE *`

`output_file` is a pointer to a previously opened file.

Constraint: The file pointer `output_file` must be opened previous to calling `fftwf_export_wisdom_to_file`.

5.9.13 _armpl_fftwf_export_wisdom_to_filname

`fftwf_export_wisdom_to_filename` exports the currently known Arm Performance Libraries FFT wisdom to a new file called `output_file` specified by the user. Note that this routine opens a new file with the supplied name, rather than *fftwf_export_wisdom_to_file* which expects a file pointer to an already open file.

Syntax

C specification:

```
#include "fftw3.h"

int fftwf_export_wisdom_to_filename(const char *filename);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

integer(C_INT) function fftwf_export_wisdom_to_filename(filename) &
    bind(C, name='fftwf_export_wisdom_to_filename')
  character(C_CHAR), dimension(*), intent(in) :: filename
end function fftwf_export_wisdom_to_filename
```

Returns

If the wisdom file is successfully created, the function returns `1`. Otherwise, the function returns `0`.

Parameters

`filename` Input parameter

`filename` is `const char *`

`filename` is a pointer to an array storing the name of the file to be opened

5.9.14 fftwf_export_wisdom_to_string

`fftwf_export_wisdom_to_string` exports the currently known Arm Performance Libraries FFT wisdom to a character string to be returned to the user.

Syntax

C specification:

```
#include "fftw3.h"

char *fftwf_export_wisdom_to_string(void);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

type(C_PTR) function fftwf_export_wisdom_to_string() &
    bind(C, name='fftwf_export_wisdom_to_string')
end function fftwf_export_wisdom_to_string
```

Returns

`fftwf_export_wisdom_to_string` returns a pointer to a string containing the previously collected wisdom.

5.9.15 fftwf_forget_wisdom

`fftwf_forget_wisdom` is intended to allow a user to discard previously loaded or learned wisdom. At present, in the Arm Performance Libraries implementation no action is taken, and the function returns no error.

Syntax

C specification:

```
#include "fftw3.h"

void fftwf_forget_wisdom(void);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

subroutine fftwf_forget_wisdom() bind(C, name='fftwf_forget_wisdom')
end subroutine fftwf_forget_wisdom
```

Returns

`fftwf_forget_wisdom` always returns 0.

5.9.16 fftwf_import_system_wisdom

`fftwf_import_system_wisdom` reads previously written Arm Performance Libraries FFT wisdom from a default system file. At present, system wisdom is unavailable for Arm Performance Libraries and this function will return 0.

Syntax

C specification:

```
#include "fftw3.h"

int fftwf_import_system_wisdom(void);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

integer(C_INT) function fftwf_import_system_wisdom() &
    bind(C, name='fftwf_import_system_wisdom')
end function fftwf_import_system_wisdom
```

Returns

`fftwf_import_system_wisdom` returns either 0 indicating that the system wisdom file was read correctly, or 1 otherwise. However since system wisdom is currently unavailable for Arm Performance Libraries, this function will return 0.

5.9.17 fftwf_import_wisdom

`fftwf_import_wisdom` calls the user-supplied function to read Arm Performance Libraries FFT wisdom in, one character at a time.

Syntax

C specification:

```
#include "fftw3.h"

int fftwf_import_wisdom(int (*read_char)(void *), void *data);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

integer(C_INT) function fftwf_import_wisdom(read_char,data) &
    bind(C, name='fftwf_import_wisdom')
  type(C_FUNPTR), value :: read_char
  type(C_PTR), value :: data
end function fftwf_import_wisdom
```

Returns

`fftwf_import_wisdom` returns either 0 indicating that the wisdom was read correctly, or 1 otherwise.

Parameters

`read_char` Input parameter

`read_char` is a pointer to a function that will read in the wisdom, one character at a time

Constraint: `write_char` takes two parameters:

the first is a `char` which is the character to be read

the second is a pointer to where this data should be read. Examples of using this would be a `FILE*` or an array

`data` Input parameter

`data` is a `void * pointer that is itself passed as the second parameter to the function ``read_char`

5.9.18 fftwf_import_wisdom_from_file

`fftwf_import_wisdom_from_file` reads the previously written Arm Performance Libraries FFT wisdom from the file open as `input_file`.

Syntax

C specification:

```
#include "fftw3.h"

int fftwf_import_wisdom_from_file(FILE *input_file);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

integer(C_INT) function fftwf_import_wisdom_from_file(input_file) &
    bind(C, name='fftwf_import_wisdom_from_file')
  type(C_PTR), value :: input_file
end function fftwf_import_wisdom_from_file
```

Returns

`fftwf_import_wisdom_from_file` returns either 0 indicating that the wisdom file was read correctly, or 1 otherwise

Parameters

input_file Input parameter

input_file is FILE *

input_file is a pointer to a previously created a file of Arm Performance Libraries FFT wisdom

5.9.19 fftwf_import_wisdom_from_filename

fftwf_import_wisdom_from_filename reads the previously written Arm Performance Libraries FFT wisdom from a file called input_file.

Syntax

C specification:

```
#include "fftw3.h"

int fftwf_import_wisdom_from_filename(const char *filename);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

integer(C_INT) function fftwf_import_wisdom_from_filename(filename) &
    bind(C, name='fftwf_import_wisdom_from_filename')
  character(C_CHAR), dimension(*), intent(in) :: filename
end function fftwf_import_wisdom_from_filename
```

Returns

fftwf_import_wisdom_from_filename returns either 0 indicating that the wisdom file was read correctly, or 1 otherwise

Parameters

filename Input parameter

filename is const char *

filename provides the name of the file from which wisdom will be loaded

5.9.20 fftwf_import_wisdom_from_string

fftwf_import_wisdom_from_string reads the previously written Arm Performance Libraries FFT wisdom from a string given by input_string.

Syntax

C specification:

```
#include "fftw3.h"

int fftwf_import_wisdom_from_string(const char *input_string);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

integer(C_INT) function fftwf_import_wisdom_from_string(input_string) &
    bind(C, name='fftwf_import_wisdom_from_string')
  character(C_CHAR), dimension(*), intent(in) :: input_string
end function fftwf_import_wisdom_from_string
```

Returns

`fftwf_import_wisdom_from_string` returns either 0 indicating that the wisdom file was read correctly, or 1 otherwise

Parameters

`input_string` Input parameter

`input_string` is `const char *`

`input_string` is a pointer to a previously created string from which the accumulated wisdom will be read

5.9.21 fftwh_export_wisdom

`fftwh_export_wisdom` calls the user-supplied function to write Arm Performance Libraries FFT wisdom out, one character at a time.

Syntax

C specification:

```
#include "fftw3.h"

void fftwh_export_wisdom(void (*write_char)(char c, void *), void *data);
```

Parameters

`write_char` Input parameter

`write_char` is a pointer to a function that will write out the wisdom accumulated by the program, one character at a time

Constraint: `write_char` takes two parameters:

the first is a `char` which is the character to be written

the second is a pointer to where this data is to be written. Examples of using this would be a `FILE*`, an array or unused and just writing to the screen

`data` Input parameter

`data` is a `void *` pointer that is itself passed as the second parameter to the function `write_char`

5.9.22 fftwh_export_wisdom_to_file

`fftwh_export_wisdom_to_file` exports the currently known Arm Performance Libraries FFT wisdom to the file opened as `output_file` specified by the user. Note that this routine expects a file pointer to an already open file, rather than *_armpl_fftwh_export_wisdom_to_filname* which opens a new file with the supplied name.

Syntax

C specification:

```
#include "fftw3.h"

void fftwh_export_wisdom_to_file(FILE *output_file);
```

Parameters

`output_file` Input parameter

`output_file` is `FILE *`

`output_file` is a pointer to a previously opened file.

Constraint: The file pointer `output_file` must be opened previous to calling `fftwh_export_wisdom_to_file`.

5.9.23 _armpl_fftwh_export_wisdom_to_filname

`fftwh_export_wisdom_to_filename` exports the currently known Arm Performance Libraries FFT wisdom to a new file called `output_file` specified by the user. Note that this routine opens a new file with the supplied name, rather than *fftwh_export_wisdom_to_file* which expects a file pointer to an already open file.

Syntax

C specification:

```
#include "fftw3.h"

int fftwh_export_wisdom_to_filename(const char *filename);
```

Returns

If the wisdom file is successfully created, the function returns `1`. Otherwise, the function returns `0`.

Parameters

`filename` Input parameter

`filename` is `const char *`

`filename` is a pointer to an array storing the name of the file to be opened

5.9.24 fftwh_export_wisdom_to_string

`fftwh_export_wisdom_to_string` exports the currently known Arm Performance Libraries FFT wisdom to a character string to be returned to the user.

Syntax

C specification:

```
#include "fftw3.h"

char *fftwh_export_wisdom_to_string(void);
```

Returns

`fftwh_export_wisdom_to_string` returns a pointer to a string containing the previously collected wisdom.

5.9.25 fftwh_forget_wisdom

`fftwh_forget_wisdom` is intended to allow a user to discard previously loaded or learned wisdom. At present, in the Arm Performance Libraries implementation no action is taken, and the function returns no error.

Syntax

C specification:

```
#include "fftw3.h"

void fftwh_forget_wisdom(void);
```

Returns

`fftwh_forget_wisdom` always returns 0.

5.9.26 fftwh_import_system_wisdom

`fftwh_import_system_wisdom` reads previously written Arm Performance Libraries FFT wisdom from a default system file. At present, system wisdom is unavailable for Arm Performance Libraries and this function will return 0.

Syntax

C specification:

```
#include "fftw3.h"

int fftwh_import_system_wisdom(void);
```

Returns

`fftwh_import_system_wisdom` returns either 0 indicating that the system wisdom file was read correctly, or 1 otherwise. However since system wisdom is currently unavailable for Arm Performance Libraries, this function will return 0.

5.9.27 fftwh_import_wisdom

`fftwh_import_wisdom` calls the user-supplied function to read Arm Performance Libraries FFT wisdom in, one character at a time.

Syntax

C specification:

```
#include "fftw3.h"

int fftwh_import_wisdom(int (*read_char)(void *), void *data);
```

Returns

`fftwh_import_wisdom` returns either 0 indicating that the wisdom was read correctly, or 1 otherwise.

Parameters

`read_char` Input parameter

`read_char` is a pointer to a function that will read in the wisdom, one character at a time

Constraint: `write_char` takes two parameters:

the first is a `char` which is the character to be read

the second is a pointer to where this data should be read. Examples of using this would be a `FILE*` or an array

`data` Input parameter

`data` is a `void * pointer that is itself passed as the second parameter to the function ``read_char`

5.9.28 fftwh_import_wisdom_from_file

`fftwh_import_wisdom_from_file` reads the previously written Arm Performance Libraries FFT wisdom from the file open as `input_file`.

Syntax

C specification:

```
#include "fftw3.h"

int fftwh_import_wisdom_from_file(FILE *input_file);
```

Returns

`fftwh_import_wisdom_from_file` returns either 0 indicating that the wisdom file was read correctly, or 1 otherwise

Parameters

input_file Input parameter

input_file is FILE *

input_file is a pointer to a previously created a file of Arm Performance Libraries FFT wisdom

5.9.29 fftwh_import_wisdom_from_filename

fftwh_import_wisdom_from_filename reads the previously written Arm Performance Libraries FFT wisdom from a file called input_file.

Syntax

C specification:

```
#include "fftw3.h"

int fftwh_import_wisdom_from_filename(const char *filename);
```

Returns

fftwh_import_wisdom_from_filename returns either 0 indicating that the wisdom file was read correctly, or 1 otherwise

Parameters

filename Input parameter

filename is const char *

filename provides the name of the file from which wisdom will be loaded

5.9.30 fftwh_import_wisdom_from_string

fftwh_import_wisdom_from_string reads the previously written Arm Performance Libraries FFT wisdom from a string given by input_string.

Syntax

C specification:

```
#include "fftw3.h"

int fftwh_import_wisdom_from_string(const char *input_string);
```

Returns

fftwh_import_wisdom_from_string returns either 0 indicating that the wisdom file was read correctly, or 1 otherwise

Parameters

input_string Input parameter

`input_string` is `const char *`

`input_string` is a pointer to a previously created string from which the accumulated wisdom will be read

5.10 FFT threading functions

5.10.1 fftw_cleanup_threads

`fftw_cleanup_threads` is intended to allow a user to discard memory and wisdom from running threaded plans previously. In the Arm Performance Libraries implementation no action is taken, since users will be using OpenMP for threading. Wisdom is stored such that the thread-count used does not invalidated wisdom with different numbers of threads.

Syntax

C specification:

```
#include "fftw3.h"

void fftw_cleanup_threads(void);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

subroutine fftw_cleanup_threads() bind(C, name='fftw_cleanup_threads')
end subroutine fftw_cleanup_threads
```

5.10.2 fftw_init_threads

`fftw_init_threads` is intended to allow a user to initialise the use of threading in the execution of FFTs. In the Arm Performance Libraries implementation threads are always ready for use, provided a suitable variety has been linked in when the application was compiled. This function, therefore takes no further action, and the function returns no error.

Syntax

C specification:

```
#include "fftw3.h"

int fftw_init_threads(void);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

integer(C_INT) function fftw_init_threads() &
    bind(C, name='fftw_init_threads')
end function fftw_init_threads
```

Returns

`fftw_init_threads` always returns 1.

5.10.3 fftw_make_planner_thread_safe

`fftw_make_planner_thread_safe` is unnecessary in the Arm Performance Libraries implementation since all planning is thread-safe.

Syntax

C specification:

```
#include "fftw3.h"

void fftw_make_planner_thread_safe();
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

subroutine fftw_make_planner_thread_safe() &
    bind(C, name='fftw_make_planner_thread_safe')
end subroutine fftw_make_planner_thread_safe
```

5.10.4 fftw_plan_with_nthreads

`fftw_plan_with_nthreads` is intended to allow a user to specify the number of threads used for the execution of FFTs. In the Arm Performance Libraries implementation threads are always ready for use, provided a suitable variety has been linked in when the application was compiled. The number of threads to be used is controlled through the usual OpenMP commands. At present this function takes no further control of setting thread counts.

Syntax

C specification:

```
#include "fftw3.h"

void fftw_plan_with_nthreads(int nthreads);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

subroutine fftw_plan_with_nthreads(nthreads) &
    bind(C, name='fftw_plan_with_nthreads')
  integer(C_INT), value :: nthreads
end subroutine fftw_plan_with_nthreads
```

Parameters

`nthreads` Input parameter

`nthreads` is `int`

This is the number of threads requested to be used when executing subsequent FFT plans

Constraint: `nthreads` >= 1

Note: This value is currently ignored

5.10.5 fftwf_cleanup_threads

`fftwf_cleanup_threads` is intended to allow a user to discard memory and wisdom from running threaded plans previously. In the Arm Performance Libraries implementation no action is taken, since users will be using OpenMP for threading. Wisdom is stored such that the thread-count used does not invalidated wisdom with different numbers of threads.

Syntax

C specification:

```
#include "fftw3.h"

void fftwf_cleanup_threads(void);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

subroutine fftwf_cleanup_threads() bind(C, name='fftwf_cleanup_threads')
end subroutine fftwf_cleanup_threads
```

5.10.6 fftwf_init_threads

`fftwf_init_threads` is intended to allow a user to initialise the use of threading in the execution of FFTs. In the Arm Performance Libraries implementation threads are always ready for use, provided a suitable variety has been linked in when the application was compiled. This function, therefore takes no further action, and the function returns no error.

Syntax

C specification:

```
#include "fftw3.h"

int fftwf_init_threads(void);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

integer(C_INT) function fftwf_init_threads() &
    bind(C, name='fftwf_init_threads')
end function fftwf_init_threads
```

Returns

`fftwf_init_threads` always returns 1.

5.10.7 fftwf_make_planner_thread_safe

`fftwf_make_planner_thread_safe` is unnecessary in the Arm Performance Libraries implementation since all planning is thread-safe.

Syntax

C specification:

```
#include "fftw3.h"

void fftwf_make_planner_thread_safe();
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

subroutine fftwf_make_planner_thread_safe() &
    bind(C, name='fftwf_make_planner_thread_safe')
end subroutine fftwf_make_planner_thread_safe
```

5.10.8 fftwf_plan_with_nthreads

`fftwf_plan_with_nthreads` is intended to allow a user to specify the number of threads used for the execution of FFTs. In the Arm Performance Libraries implementation threads are always ready for use, provided a suitable variety has been linked in when the application was compiled. The number of threads to be used is controlled through the usual OpenMP commands. At present this function takes no further control of setting thread counts.

Syntax

C specification:

```
#include "fftw3.h"

void fftwf_plan_with_nthreads(int nthreads);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

subroutine fftwf_plan_with_nthreads(nthreads) &
    bind(C, name='fftwf_plan_with_nthreads')
  integer(C_INT), value :: nthreads
end subroutine fftwf_plan_with_nthreads
```

Parameters

nthreads Input parameter

`nthreads` is `int`

This is the number of threads requested to be used when executing subsequent FFT plans

Constraint: `nthreads` >= 1

Note: This value is currently ignored

5.10.9 fftwh_cleanup_threads

`fftwh_cleanup_threads` is intended to allow a user to discard memory and wisdom from running threaded plans previously. In the Arm Performance Libraries implementation no action is taken, since users will be using OpenMP for threading. Wisdom is stored such that the thread-count used does not invalidated wisdom with different numbers of threads.

Syntax

C specification:

```
#include "fftw3.h"

void fftwh_cleanup_threads(void);
```

5.10.10 fftwh_init_threads

`fftwh_init_threads` is intended to allow a user to initialise the use of threading in the execution of FFTs. In the Arm Performance Libraries implementation threads are always ready for use, provided a suitable variety has been linked in when the application was compiled. This function, therefore takes no further action, and the function returns no error.

Syntax

C specification:

```
#include "fftw3.h"

int fftwh_init_threads(void);
```

Returns

`fftwh_init_threads` always returns 1.

5.10.11 fftwh_make_planner_thread_safe

`fftwh_make_planner_thread_safe` is unnecessary in the Arm Performance Libraries implementation since all planning is thread-safe.

Syntax

C specification:

```
#include "fftw3.h"

void fftwh_make_planner_thread_safe();
```

5.10.12 fftwh_plan_with_nthreads

`fftwh_plan_with_nthreads` is intended to allow a user to specify the number of threads used for the execution of FFTs. In the Arm Performance Libraries implementation threads are always ready for use, provided a suitable variety has been linked in when the application was compiled. The number of threads to be used is controlled through the usual OpenMP commands. At present this function takes no further control of setting thread counts.

Syntax

C specification:

```
#include "fftw3.h"

void fftwh_plan_with_nthreads(int nthreads);
```

Parameters

`nthreads` Input parameter

`nthreads` is `int`

This is the number of threads requested to be used when executing subsequent FFT plans

Constraint: `nthreads` >= 1

Note: This value is currently ignored

5.11 FFT utilities functions

5.11.1 fftw_cost

`fftw_cost` provides a measure of the estimated work in executing an FFT plan. In Arm Performance Libraries all costs are currently returned as 0.0.

Syntax

C specification:

```
#include "fftw3.h"

double fftw_cost(const fftw_plan p);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

real(C_DOUBLE) function fftw_cost(p) bind(C, name='fftw_cost')
  type(C_PTR), value :: p
end function fftw_cost
```

Returns

`fftw_cost` always returns 0.0.

Parameters

p Input parameter

`p` is `fftw_plan`

An FFT plan created from a preceding call

5.11.2 fftw_estimate_cost

`fftw_estimate_cost` provides a measure of the estimated work in executing an FFT plan. In Arm Performance Libraries all costs are currently returned as 0.0.

Syntax

C specification:

```
#include "fftw3.h"

double fftw_estimate_cost(const fftw_plan p);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

real(C_DOUBLE) function fftw_estimate_cost(p) &
    bind(C, name='fftw_estimate_cost')
  type(C_PTR), value :: p
end function fftw_estimate_cost
```

Returns

`fftw_estimate_cost` always returns 0.0.

Parameters

p Input parameter

`p` is `fftw_plan`

An FFT plan created from a preceding call

5.11.3 fftw_flops

`fftw_flops` provides a measure of the number of adds, multiplies and FMAs done during execution of an FFT plan. In Arm Performance Libraries all costs are currently returned as 0.0.

Syntax

C specification:

```
#include "fftw3.h"

void fftw_flops(const fftw_plan plan, double *add, double *mul, double *fma);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

subroutine fftw_flops(p,add,mul,fmas) bind(C, name='fftw_flops')
  type(C_PTR), value :: p
  real(C_DOUBLE), intent(out) :: add
  real(C_DOUBLE), intent(out) :: mul
  real(C_DOUBLE), intent(out) :: fmas
end subroutine fftw_flops

real(C_DOUBLE) function fftw_estimate_cost(p) &
    bind(C, name='fftw_estimate_cost')
  type(C_PTR), value :: p
end function fftw_estimate_cost
```

Parameters

p Input parameter

`p` is `fftw_plan`

An FFT plan created from a preceding call

add Input parameter

`add` is a `double*`

`add` is a pointer to a single `double` value into which the number of adds is returned

mul Input parameter

`mul` is a `double*`

`mul` is a pointer to a single `double` value into which the number of multiplies is returned

fma Input parameter

`fma` is a `double*`

`fma` is a pointer to a single `double` value into which the number of FMAs is returned

5.11.4 fftw_fprint_plan

`fftw_fprint_plan` is indended to provide a method of printing an FFT plan to a previously opened file. In Arm Performance Libraries this functionality is currently not supported, and a warning message will be printed to STDERR instead.

Syntax

C specification:

```
#include "fftw3.h"

void fftw_fprint_plan(const fftw_plan plan, FILE *out);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

subroutine fftw_fprint_plan(p,output_file) bind(C, name='fftw_fprint_plan')
  type(C_PTR), value :: p
  type(C_PTR), value :: output_file
end subroutine fftw_fprint_plan
```

Parameters

p Input parameter

p is fftw_plan

An FFT plan created from a preceding call

out Input parameter

out is FILE *

out is a pointer to a previously opened file.

Constraint: The file pointer output_file must be opened previous to calling fftw_fprintf_plan.

5.11.5 fftw_print_plan

fftw_print_plan is indended to provide a method of printing an FFT plan to STDOUT. In Arm Performance Libraries this functionality is currently not supported, and a warning message will be printed to STDERR instead.

Syntax

C specification:

```
#include "fftw3.h"

void fftw_print_plan(const fftw_plan p);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

subroutine fftw_print_plan(p) bind(C, name='fftw_print_plan')
  type(C_PTR), value :: p
end subroutine fftw_print_plan
```

Parameters

p Input parameter

p is fftw_plan

An FFT plan created from a preceding call

5.11.6 fftw_set_timelimit

fftw_set_timelimit sets a recommended maximum time that should be used in planning FFT transforms. The Arm Performance Libraries implementation uses this as a guideline, stopping once the allowed time has been exhausted, or if it expects further planning to take it over this limit. The use of planning flags, e.g. FFTW_EXHAUSTIVE and friends, are all truncated at this time limit.

To unset the timelimit pass in FFTW_NO_TIMELIMIT to seconds.

Syntax

C specification:

```
#include "fftw3.h"

void fftw_set_timelimit(double seconds);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

subroutine fftw_set_timelimit(t) bind(C, name='fftw_set_timelimit')
  real(C_DOUBLE), value :: t
end subroutine fftw_set_timelimit
```

Parameters

seconds Input parameter

seconds is a double

seconds is a the maximum number of seconds for which the planning time for subsequent FFT plans will be limited.

Note: Pass in FFTW_NO_TIMELIMIT to unset this limit

5.11.7 fftwf_cost

fftwf_cost provides a measure of the estimated work in executing an FFT plan. In Arm Performance Libraries all costs are currently returned as 0.0.

Syntax

C specification:

```
#include "fftw3.h"

double fftwf_cost(const fftwf_plan p);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

real(C_DOUBLE) function fftwf_cost(p) bind(C, name='fftwf_cost')
  type(C_PTR), value :: p
end function fftwf_cost
```

Returns

`fftwf_cost` always returns 0.0.

Parameters

p Input parameter

`p` is `fftwf_plan`

An FFT plan created from a preceding call

5.11.8 fftwf_estimate_cost

`fftwf_estimate_cost` provides a measure of the estimated work in executing an FFT plan. In Arm Performance Libraries all costs are currently returned as 0.0.

Syntax

C specification:

```
#include "fftw3.h"

double fftwf_estimate_cost(const fftwf_plan p);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

real(C_DOUBLE) function fftwf_estimate_cost(p) bind(C, name='fftwf_estimate_cost')
  type(C_PTR), value :: p
end function fftwf_estimate_cost
```

Returns

`fftwf_estimate_cost` always returns 0.0.

Parameters

p Input parameter

`p` is `fftwf_plan`

An FFT plan created from a preceding call

5.11.9 fftwf_flops

`fftwf_flops` provides a measure of the number of adds, multiplies and FMAs done during execution of an FFT plan. In Arm Performance Libraries all costs are currently returned as 0.0.

Syntax

C specification:

```
#include "fftw3.h"

void fftwf_flops(const fftwf_plan plan, double *add, double *mul, double *fma);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

subroutine fftwf_flops(p,add,mul,fmas) bind(C, name='fftwf_flops')
  type(C_PTR), value :: p
  real(C_DOUBLE), intent(out) :: add
  real(C_DOUBLE), intent(out) :: mul
  real(C_DOUBLE), intent(out) :: fmas
end subroutine fftwf_flops

real(C_DOUBLE) function fftwf_estimate_cost(p) &
    bind(C, name='fftwf_estimate_cost')
  type(C_PTR), value :: p
end function fftwf_estimate_cost
```

Parameters

p Input parameter

`p` is `fftwf_plan`

An FFT plan created from a preceding call

add Input parameter

`add` is a `double*`

`add` is a pointer to a single `double` value into which the number of adds is returned

mul Input parameter

`mul` is a `double*`

`mul` is a pointer to a single `double` value into which the number of multiplies is returned

fma Input parameter

`fma` is a `double*`

`fma` is a pointer to a single `double` value into which the number of FMAs is returned

5.11.10 fftwf_fprint_plan

`fftwf_fprint_plan` is indended to provide a method of printing an FFT plan to a previously opened file. In Arm Performance Libraries this functionality is currently not supported, and a warning message will be printed to STDERR instead.

Syntax

C specification:

```
#include "fftw3.h"

void fftwf_fprint_plan(const fftwf_plan plan, FILE *out);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

subroutine fftwf_fprint_plan(p,output_file) &
    bind(C, name='fftwf_fprint_plan')
  type(C_PTR), value :: p
  type(C_PTR), value :: output_file
end subroutine fftwf_fprint_plan
```

Parameters

p Input parameter

p is fftwf_plan

An FFT plan created from a preceding call

out Input parameter

out is FILE *

out is a pointer to a previously opened file.

Constraint: The file pointer output_file must be opened previous to calling fftwf_fprintf_plan.

5.11.11 fftwf_print_plan

fftwf_print_plan is indended to provide a method of printing an FFT plan to STDOUT. In Arm Performance Libraries this functionality is currently not supported, and a warning message will be printed to STDERR instead.

Syntax

C specification:

```
#include "fftw3.h"

void fftwf_print_plan(const fftwf_plan p);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

subroutine fftwf_print_plan(p) bind(C, name='fftwf_print_plan')
  type(C_PTR), value :: p
end subroutine fftwf_print_plan
```

Parameters

p Input parameter

p is fftwf_plan

An FFT plan created from a preceding call

5.11.12 fftwf_set_timelimit

fftwf_set_timelimit sets a recommended maximum time that should be used in planning FFT transforms. The Arm Performance Libraries implementation uses this as a guideline, stopping once the allowed time has been exhausted, or if it expects further planning to take it over this limit. The use of planning flags, e.g. FFTW_EXHAUSTIVE and friends, are all truncated at this time limit.

To unset the timelimit pass in FFTW_NO_TIMELIMIT to seconds.

Syntax

C specification:

```
#include "fftw3.h"

void fftwf_set_timelimit(double seconds);
```

Fortran 2003 specification:

```
use, intrinsic :: iso_c_binding
include 'fftw3.f03'

subroutine fftwf_set_timelimit(t) bind(C, name='fftwf_set_timelimit')
  real(C_DOUBLE), value :: t
end subroutine fftwf_set_timelimit
```

Parameters

seconds Input parameter

seconds is a double

seconds is a the maximum number of seconds for which the planning time for subsequent FFT plans will be limited.

Note: Pass in FFTW_NO_TIMELIMIT to unset this limit

5.11.13 fftwh_cost

fftwh_cost provides a measure of the estimated work in executing an FFT plan. In Arm Performance Libraries all costs are currently returned as 0.0.

Syntax

C specification:

```
#include "fftw3.h"

double fftwh_cost(const fftwh_plan p);
```

Returns

`fftwh_cost` always returns 0.0.

Parameters

p Input parameter

`p` is `fftwh_plan`

An FFT plan created from a preceding call

5.11.14 fftwh_estimate_cost

`fftwh_estimate_cost` provides a measure of the estimated work in executing an FFT plan. In Arm Performance Libraries all costs are currently returned as 0.0.

Syntax

C specification:

```
#include "fftw3.h"

double fftwh_estimate_cost(const fftwh_plan p);
```

Returns

`fftwh_estimate_cost` always returns 0.0.

Parameters

p Input parameter

`p` is `fftwh_plan`

An FFT plan created from a preceding call

5.11.15 fftwh_flops

`fftwh_flops` provides a measure of the number of adds, multiplies and FMAs done during execution of an FFT plan. In Arm Performance Libraries all costs are currently returned as 0.0.

Syntax

C specification:

```
#include "fftw3.h"

void fftwh_flops(const fftwh_plan plan, double *add, double *mul, double *fma);
```

Parameters

p Input parameter

p is fftwh_plan

An FFT plan created from a preceding call

add Input parameter

add is a double*

add is a pointer to a single double value into which the number of adds is returned

mul Input parameter

mul is a double*

mul is a pointer to a single double value into which the number of multiplies is returned

fma Input parameter

fma is a double*

fma is a pointer to a single double value into which the number of FMAs is returned

5.11.16 fftwh_fprint_plan

fftwh_fprint_plan is indended to provide a method of printing an FFT plan to a previously opened file. In Arm Performance Libraries this functionality is currently not supported, and a warning message will be printed to STDERR instead.

Syntax

C specification:

```
#include "fftw3.h"

void fftwh_fprint_plan(const fftwh_plan plan, FILE *out);
```

Parameters

p Input parameter

p is fftwh_plan

An FFT plan created from a preceding call

out Input parameter

out is FILE *

out is a pointer to a previously opened file.

Constraint: The file pointer output_file must be opened previous to calling fftwh_fprintf_plan.

5.11.17 fftwh_print_plan

fftwh_print_plan is indended to provide a method of printing an FFT plan to STDOUT. In Arm Performance Libraries this functionality is currently not supported, and a warning message will be printed to STDERR instead.

Syntax

C specification:

```
#include "fftw3.h"

void fftwh_print_plan(const fftwh_plan p);
```

Parameters

p Input parameter

`p` is `fftwh_plan`

An FFT plan created from a preceding call

5.11.18 fftwh_set_timelimit

`fftwh_set_timelimit` sets a recommended maximum time that should be used in planning FFT transforms. The Arm Performance Libraries implementation uses this as a guideline, stopping once the allowed time has been exhausted, or if it expects further planning to take it over this limit. The use of planning flags, e.g. `FFTW_EXHAUSTIVE` and friends, are all truncated at this time limit.

To unset the timelimit pass in `FFTW_NO_TIMELIMIT` to `seconds`.

Syntax

C specification:

```
#include "fftw3.h"

void fftwh_set_timelimit(double seconds);
```

Parameters

seconds Input parameter

`seconds` is a `double`

`seconds` is a the maximum number of seconds for which the planning time for subsequent FFT plans will be limited.

Note: Pass in `FFTW_NO_TIMELIMIT` to unset this limit

CHAPTER

SIX

SPARSE LINEAR ALGEBRA

6.1 Sparse Linear Algebra Introduction

Vendor libraries, like Arm Performance Libraries, aim to provide the highest performing linear algebra solutions, and significant gains in performance are possible over a reference version through using a tuned implementation. Extending this model to sparse linear algebra does give users the option of performance improvements through use of the vendor library implementation. However, unlike for BLAS and LAPACK, there is no standard interface for sparse linear algebra. Instead we have had to adopt our own interface that offers users the opportunity for the largest performance gains, and is not dissimilar to other vendor libraries offerings.

The implementation of sparse linear algebra in Arm Performance Libraries uses an *inspector-executor* model. This allows users to create the sparse matrix structure once and reuse it many times, and is based around providing performance for this very common use-case. There is therefore a performance overhead during the first *optimize* phase, but repeated calls to *execute* will give noticeable benefits in performance.

If the structure is only going to be used once the user can provide a *hint* to the library to minimize the amount of preprocessing done on this data, ensuring that single-shot execution is not slowed down by creating structure that would only give benefits with many executions. Additional hints can be given to help the library make further optimizations including information on the structure (e.g. symmmetric, banded, etc), usage (e.g. transposed) and memory (e.g. whether allocating should be allowed). Note these hints might be used, or ignored, as appropriate during the creation of the internal sparse representation of the matrix.

The basic usage pattern is given by the following sequence for an SpMV use-case:

1. Create sparse matrix object: `armpl_spmat_create_csr_[sdcz]()`
2. Supply hints on usage: `armpl_spmat_hint()`
3. Optimize for SpMV: `armpl_spmv_optimize()`
4. Solve SpMV case: `armpl_spmv_solve_[sdcz]()`
5. Destroy sparse matrix object: `armpl_spmat_destroy()`

In addition users can choose to update a set of non-zero values using `armpl_spmat_update_[sdcz]()`.

Each routine returns an error handle that can be used to interrogate errors during execution.

Note also, at present this interface is only available from the C interface. A Fortran interface will be added in a future release.

6.2 Example of SpMV usage

An example case of using the Arm Performance Libraries sparse interface is given through the following code.

```
/* Double precision sparse matrix-vector multiplication example */
/*
 * ARMPL version 19.2 Copyright Arm 2019
 */
```

(continues on next page)

(continued from previous page)

```
#include <stdio.h>
#include <stdlib.h>
#include "armpl.h"

#define NNZ 12
#define M 5
#define N 5

int main()
{
    /* 1. Set-up local CSR structure */
    armpl_spmat_t armpl_mat;
    const armpl_int_t ntests = 1000;
    const double alpha = 1.0, beta = 0.0;
    armpl_int_t creation_flags = 0;
    double vals[NNZ] = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0,
                        11.0, 12.0};
    armpl_int_t row_ptr[M+1] = {0, 2, 4, 7, 9, 12};
    armpl_int_t col_indx[NNZ] = {0, 2, 1, 3, 1, 2, 3, 2, 3, 3, 4};

    /* 2. Set-up Arm Performance Libraries sparse matrix object */
    armpl_status_t info = armpl_spmat_create_csr_d(&armpl_mat, M, N, row_ptr,
                                col_indx, vals, creation_flags);
    if (info!=ARMPL_STATUS_SUCCESS)
          printf("ERROR: armpl_spmat_create_csr_d returned %d\n", info);

    /* 3a. Supply any pertinent information that is known about the matrix */
    info = armpl_spmat_hint(armpl_mat, ARMPL_SPARSE_HINT_STRUCTURE,
                            ARMPL_SPARSE_STRUCTURE_UNSTRUCTURED);
    if (info!=ARMPL_STATUS_SUCCESS)
          printf("ERROR: armpl_spmat_hint returned %d\n", info);

    /* 3b. Supply any hints that are about the SpMV calculations
         to be performed */
    info = armpl_spmat_hint(armpl_mat, ARMPL_SPARSE_HINT_SPMV_OPERATION,
                            ARMPL_SPARSE_OPERATION_NOTRANS);
    if (info!=ARMPL_STATUS_SUCCESS)
          printf("ERROR: armpl_spmat_hint returned %d\n", info);

    info = armpl_spmat_hint(armpl_mat, ARMPL_SPARSE_HINT_SPMV_INVOCATIONS,
                            ARMPL_SPARSE_INVOCATIONS_MANY);
    if (info!=ARMPL_STATUS_SUCCESS)
          printf("ERROR: armpl_spmat_hint returned %d\n", info);

    /* 4. Call an optimization process that will learn from the hints you
        have previously supplied */
    info = armpl_spmv_optimize(armpl_mat);
    if (info!=ARMPL_STATUS_SUCCESS)
          printf("ERROR: armpl_spmv_optimize returned %d\n", info);

    /* 5. Setup input and output vectors and then do SpMV and print result. */
    double *x = (double *)malloc(N*sizeof(double));
    for (int i=0; i<N; i++) {
            x[i] = 1.0;
    }
    double *y = (double *)malloc(M*sizeof(double));

    for (int i=0; i<ntests; i++) {
            info = armpl_spmv_exec_d(ARMPL_SPARSE_OPERATION_NOTRANS, alpha,
                            armpl_mat, x, beta, y);
```

(continues on next page)

(continued from previous page)

```
            if (info!=ARMPL_STATUS_SUCCESS)
            printf("ERROR: armpl_spmv_exec_d returned %d\n", info);
    }

    printf("Computed vector y:\n");
    for (int i=0; i<M; i++) {
            printf("\t%2.1f\n", y[i]);
    }

    /* 6. Destroy created matrix to free any memory created during the
        'optimize' phase */
    info = armpl_spmat_destroy(armpl_mat);
    if (info!=ARMPL_STATUS_SUCCESS)
          printf("ERROR: armpl_spmat_destroy returned %d\n", info);

    /* 7. Free user allocated storage */
    free(x); free(y);

    return (int)info;
}
```

6.3 Sparse Linear Algebra Functions

6.3.1 armpl_spmat_create_coo_c

To create a complex single precision sparse matrix object from input data given in COOrdinate (COO) format, use **armpl_spmat_create_coo_c**.

Syntax

C specification:

```
#include "armpl.h"

armpl_status_t armpl_spmat_create_coo_c(armpl_spmat_t *A, armpl_int_t m,
                                        armpl_int_t n, armpl_int_t nnz,
                                        const armpl_int_t *row_indx,
                                        const armpl_int_t *col_indx,
                                        const armpl_singlecomplex_t *vals,
                                        armpl_int_t flags);
```

Fortran 2003 specification:

```
use armpl_library

subroutine armpl_spmat_create_coo(A, m, n, nnz, row_indx, col_indx, vals, flags,␣
↪info)
         use armpl_kinds
         integer(kind=armpl_i8), intent(out) :: A
         integer, intent(in) :: m, n, nnz
         integer, intent(in) :: row_indx(*)
         integer, intent(in) :: col_indx(*)
         real(kind=armpl_r64), intent(in) :: vals(*)
         integer, intent(in) :: flags
         integer, intent(out) :: info
end subroutine armpl_spmat_create_coo
```

Returns

In C/C++, this function returns an `armpl_status_t`, which can be tested to check that the function returned correctly. In Fortran, this value is stored in the parameter `info`.

Possible return values are:

- **`ARMPL_STATUS_SUCCESS`** The function returned with no errors.
- **`ARMPL_STATUS_INPUT_PARAMETER_ERROR`** An error was found with the supplied input parameters.
- **`ARMPL_STATUS_EXECUTION_FAILURE`** The function failed for an alternative reason.

Parameters

`A` Output parameter

`A` is `armpl_spmat_t*`

The sparse matrix object to be created.

`m` Input parameter

`m` is `armpl_int_t`

`m` is the number of rows in the sparse matrix.

Constraint: `m` >= 0

`n` Input parameter

`n` is `armpl_int_t`

`n` is the number of columns in the sparse matrix.

Constraint: `n` >= 0

`nnz` Input parameter

`nnz` is `armpl_int_t`

`nnz` is the number of non-zeroes in the sparse matrix.

Constraint: `nnz` >= 0

`row_indx` Input parameter

`row_indx` is `const armpl_int_t*`

`row_indx` is an array listing the rows for each non-zero entry.

Constraints: Array of length `m`.

Although not checked at runtime, `col_indx` are expected to be in the range:

- Using the C/C++ interface: 0 <= `row_indx[i]` <= `n-1`.
- Using the Fortran interface: 1 <= `row_indx[i]` <= `n`.

`col_indx` Input parameter

`col_indx` is `const armpl_int_t*`

`col_indx` is an array listing the columns for each non-zero entry.

Constraints: Array of length `n`.

Although not checked at runtime, `col_indx` are expected to be in the range:

- Using the C/C++ interface: 0 <= `col_indx[i]` <= `n-1`.
- Using the Fortran interface: 1 <= `col_indx[i]` <= `n`.

vals Input parameter

`vals` is `const armpl_singlecomplex_t*`

Non-zero entries in the matrix to be represented in the sparse matrix structure.

Constraint: Array of length `nnz`

flags Input parameter

`flags` is `armpl_int_t`

Use this flag to create certain properties of the matrix. It can be set to:

- `0`. No restrictions apply. The function will make a copy of the input arrays, which can be deallocated by the user on return from this function in order to minimize memory usage.
- `ARMPL_SPARSE_CREATE_NOCOPY`. The function will not make a copy of the input arrays. These input arrays, therefore, **must not** be deallocated until after a call to `armpl_spmat_destroy` with the `armpl_spmat` object returned by this function. This will reduce the scope of optimizations that might be applied in subsequent calls to `armpl_spmv_optimize`.

6.3.2 armpl_spmat_create_coo_d

To create a real double precision sparse matrix object from input data given in COOrdinate (COO) format, use **armpl_spmat_create_coo_d**.

Syntax

C specification:

```
#include "armpl.h"

armpl_status_t armpl_spmat_create_coo_d(armpl_spmat_t *A, armpl_int_t m,
                                        armpl_int_t n, armpl_int_t nnz,
                                        const armpl_int_t *row_indx,
                                        const armpl_int_t *col_indx,
                                        const double *vals,
                                        armpl_int_t flags);
```

Fortran 2003 specification:

```
use armpl_library

subroutine armpl_spmat_create_coo(A, m, n, nnz, row_indx, col_indx, vals, flags,␣
↪info)
         use armpl_kinds
         integer(kind=armpl_i8), intent(out) :: A
         integer, intent(in) :: m, n, nnz
         integer, intent(in) :: row_indx(*)
         integer, intent(in) :: col_indx(*)
         complex(kind=armpl_r32), intent(in) :: vals(*)
         integer, intent(in) :: flags
         integer, intent(out) :: info
end subroutine armpl_spmat_create_coo
```

Returns

In C/C++, this function returns an `armpl_status_t`, which can be tested to check that the function returned correctly. In Fortran, this value is stored in the parameter `info`.

Possible return values are:

- **ARMPL_STATUS_SUCCESS** The function returned with no errors.
- **ARMPL_STATUS_INPUT_PARAMETER_ERROR** An error was found with the supplied input parameters.
- **ARMPL_STATUS_EXECUTION_FAILURE** The function failed for an alternative reason.

Parameters

A Output parameter

A is armpl_spmat_t*

The sparse matrix object to be created.

m Input parameter

m is armpl_int_t

m is the number of rows in the sparse matrix.

Constraint: m >= 0

n Input parameter

n is armpl_int_t

n is the number of columns in the sparse matrix.

Constraint: n >= 0

nnz Input parameter

nnz is armpl_int_t

nnz is the number of non-zeroes in the sparse matrix.

Constraint: nnz >= 0

row_indx Input parameter

row_indx is const armpl_int_t*

row_indx is an array listing the rows for each non-zero entry.

Constraints: Array of length m.

Although not checked at runtime, row_indx are expected to be in the range:

- Using the C/C++ interface: 0 <= row_indx[i] <= n-1.
- Using the Fortran interface: 1 <= row_indx[i] <= n.

col_indx Input parameter

col_indx is const armpl_int_t*

col_indx is an array listing the columns for each non-zero entry.

Constraints: Array of length n

Although not checked at runtime, col_indx are expected to be in the range:

- Using the C/C++ interface: 0 <= col_indx[i] <= n-1.
- Using the Fortran interface: 1 <= col_indx[i] <= n.

vals Input parameter

vals is const double*

Non-zero entries in the matrix to be represented in the sparse matrix structure.

Constraint: Array of length nnz

flags Input parameter

`flags` is `armpl_int_t`

Use this flag to create certain properties of the matrix. This can be set to:

- `0`. No restrictions apply. The function will make a copy of the input arrays, which can be deallocated by the user on return from this function in order to minimize memory usage.
- `ARMPL_SPARSE_CREATE_NOCOPY`. The function will not make a copy of the input arrays. These input arrays, therefore, **must not** be deallocated until after a call to `armpl_spmat_destroy` with the `armpl_spmat` object returned by this function. This will reduce the scope of optimizations that might be applied in subsequent calls to `armpl_spmv_optimize`.

6.3.3 armpl_spmat_create_coo_s

To create a real single precision sparse matrix object from input data given in COOrdinate (COO) format, use **armpl_spmat_create_coo_s**.

Syntax

C specification:

```
#include "armpl.h"

armpl_status_t armpl_spmat_create_coo_s(armpl_spmat_t *A, armpl_int_t m,
                                        armpl_int_t n, armpl_int_t nnz,
                                        const armpl_int_t *row_indx,
                                        const armpl_int_t *col_indx,
                                        const float *vals,
                                        armpl_int_t flags);
```

Fortran 2003 specification:

```
use armpl_library

subroutine armpl_spmat_create_coo(A, m, n, nnz, row_indx, col_indx, vals, flags,␣
↪info)
         use armpl_kinds
         integer(kind=armpl_i8), intent(out) :: A
         integer, intent(in) :: m, n, nnz
         integer, intent(in) :: row_indx(*)
         integer, intent(in) :: col_indx(*)
         real(kind=armpl_r32), intent(in) :: vals(*)
         integer, intent(in) :: flags
         integer, intent(out) :: info
end subroutine armpl_spmat_create_coo
```

Returns

In C/C++, this function returns an `armpl_status_t`, which can be tested to check that the function returned correctly. In Fortran, this value is stored in the parameter `info`.

Possible return values are:

- **ARMPL_STATUS_SUCCESS** The function returned with no errors.
- **ARMPL_STATUS_INPUT_PARAMETER_ERROR** An error was found with the supplied input parameters.
- **ARMPL_STATUS_EXECUTION_FAILURE** The function failed for an alternative reason.

Parameters

A Output parameter

A is `armpl_spmat_t*`

The sparse matrix object to be created.

m Input parameter

m is `armpl_int_t`

m is the number of rows in the sparse matrix.

Constraint: m >= 0

n Input parameter

n is `armpl_int_t`

n is the number of columns in the sparse matrix.

Constraint: n >= 0

nnz Input parameter

nnz is `armpl_int_t`

nnz is the number of non-zeroes in the sparse matrix.

Constraint: nnz >= 0

row_indx Input parameter

row_indx is `const armpl_int_t*`

row_indx is an array listing the rows for each non-zero entry.

Constraints: Array of length m.

Although not checked at runtime, row_indx are expected to be in the range:

- Using the C/C++ interface: 0 <= `row_indx[i]` <= `n-1`.
- Using the Fortran interface: 1 <= `row_indx[i]` <= n.

col_indx Input parameter

col_indx is `const armpl_int_t*`

col_indx is an array listing the columns for each non-zero entry.

Constraints: Array of length n.

Although not checked at runtime, col_indx are expected to be in the range:

- Using the C/C++ interface: 0 <= `col_indx[i]` <= `n-1`.
- Using the Fortran interface: 1 <= `col_indx[i]` <= n.

vals Input parameter

vals is `const float*`

Non-zero entries in the matrix to be represented in the sparse matrix structure.

Constraint: Array of length nnz

flags Input parameter

flags is `armpl_int_t`

Use this flag to create certain properties of the matrix. This can be set to:

- 0. No restrictions apply. The function will make a copy of the input arrays, which can be deallocated by the user on return from this function in order to minimize memory usage.

- `ARMPL_SPARSE_CREATE_NOCOPY`. The function will not make a copy of the input arrays. These input arrays, therefore, **must not** be deallocated until after a call to `armpl_spmat_destroy` with the `armpl_spmat` object returned by this function. This will reduce the scope of optimizations that might be applied in subsequent calls to `armpl_spmv_optimize`.

6.3.4 armpl_spmat_create_coo_z

To create a complex double precision sparse matrix object from input data given in COOrdinate (COO) format, use **armpl_spmat_create_coo_z**.

Syntax

C specification:

```
#include "armpl.h"

armpl_status_t armpl_spmat_create_coo_z(armpl_spmat_t *A, armpl_int_t m,
                                        armpl_int_t n, armpl_int_t nnz,
                                        const armpl_int_t *row_indx,
                                        const armpl_int_t *col_indx,
                                        const armpl_doublecomplex_t *vals,
                                        armpl_int_t flags);
```

Fortran 2003 specification:

```
use armpl_library

subroutine armpl_spmat_create_coo(A, m, n, nnz, row_indx, col_indx, vals, flags,␣
↪info)
         use armpl_kinds
         integer(kind=armpl_i8), intent(out) :: A
         integer, intent(in) :: m, n, nnz
         integer, intent(in) :: row_indx(*)
         integer, intent(in) :: col_indx(*)
         complex(kind=armpl_r64), intent(in) :: vals(*)
         integer, intent(in) :: flags
         integer, intent(out) :: info
end subroutine armpl_spmat_create_coo
```

Returns

In C/C++, this function returns an `armpl_status_t`, which can be tested to check that the function returned correctly. In Fortran, this value is stored in the parameter `info`.

Possible return values are:

- **`ARMPL_STATUS_SUCCESS`** The function returned with no errors.
- **`ARMPL_STATUS_INPUT_PARAMETER_ERROR`** An error was found with the supplied input parameters.
- **`ARMPL_STATUS_EXECUTION_FAILURE`** The function failed for an alternative reason.

Parameters

`A` Output parameter

`A` is `armpl_spmat_t*`

The sparse matrix object to be created.

m Input parameter

m is armpl_int_t

m is the number of rows in the sparse matrix.

Constraint: m >= 0

n Input parameter

n is armpl_int_t

n is the number of columns in the sparse matrix.

Constraint: n >= 0

nnz Input parameter

nnz is armpl_int_t

nnz is the number of non-zeroes in the sparse matrix.

Constraint: nnz >= 0

row_indx Input parameter

row_indx is const armpl_int_t*

row_indx is an array listing the rows for each non-zero entry.

Constraints: Array of length m.

Although not checked at runtime, row_indx are expected to be in the range:

- Using the C/C++ interface: 0 <= row_indx[i] <= n-1.
- Using the Fortran interface: 1 <= row_indx[i] <= n.

col_indx Input parameter

col_indx is const armpl_int_t*

col_indx is an array listing the columns for each non-zero entry.

Constraints: Array of length n.

Although not checked at runtime, col_indx are expected to be in the range:

- Using the C/C++ interface: 0 <= col_indx[i] <= n-1.
- Using the Fortran interface: 1 <= col_indx[i] <= n.

vals Input parameter

vals is const armpl_doublecomplex_t*

Non-zero entries in the matrix to be represented in the sparse matrix structure.

Constraint: Array of length nnz

flags Input parameter

flags is armpl_int_t

Use this flag to create certain properties of the matrix. This can be set to:

- 0. No restrictions apply. The function will make a copy of the input arrays, which can be deallocated by the user on return from this function in order to minimize memory usage.
- ARMPL_SPARSE_CREATE_NOCOPY. The function will not make a copy of the input arrays. These input arrays, therefore, **must not** be deallocated until after a call to armpl_spmat_destroy with the armpl_spmat object returned by this function. This will reduce the scope of optimizations that might be applied in subsequent calls to armpl_spmv_optimize.

6.3.5 armpl_spmat_create_csc_c

To create a complex single precision sparse matrix object from input data given in Compressed Sparse Column (CSC) format, use **armpl_spmat_create_csc_c**.

Syntax

C specification:

```
#include "armpl.h"

armpl_status_t armpl_spmat_create_csc_c(armpl_spmat_t *A, armpl_int_t m,
                                        armpl_int_t n,
                                        const armpl_int_t *row_indx,
                                        const armpl_int_t *col_ptr,
                                        const armpl_singlecomplex_t *vals,
                                        armpl_int_t flags);
```

Fortran 2003 specification:

```
use armpl_library

subroutine armpl_spmat_create_csc(A, m, n, row_indx, col_ptr, vals, flags, info)
         use armpl_kinds
         integer(kind=armpl_i8), intent(out) :: A
         integer, intent(in) :: m, n
         integer, intent(in) :: row_indx(*)
         integer, intent(in) :: col_ptr(n+1)
         complex(kind=armpl_r32), intent(in) :: vals(*)
         integer, intent(in) :: flags
         integer, intent(out) :: info
end subroutine armpl_spmat_create_csc
```

Returns

In C/C++, this function returns an `armpl_status_t`, which can be tested to check that the function returned correctly. In Fortran, this value is stored in the parameter `info`.

Possible return values are:

- **`ARMPL_STATUS_SUCCESS`** The function returned with no errors.
- **`ARMPL_STATUS_INPUT_PARAMETER_ERROR`** An error was found with the supplied input parameters.
- **`ARMPL_STATUS_EXECUTION_FAILURE`** The function failed for an alternative reason.

Parameters

`A` Output parameter

`A` is `armpl_spmat_t*`

The sparse matrix object to be created.

`m` Input parameter

`m` is `armpl_int_t`

`m` is the number of rows in the sparse matrix.

Constraint: `m` >= 0

n Input parameter

n is armpl_int_t

n is the number of columns in the sparse matrix.

Constraint: n >= 0

row_indx Input parameter

row_indx is const armpl_int_t*

row_indx is an array listing the rows for each non-zero entry.

Constraints: Array of length NNZ = col_ptr[M] - col_ptr[0] (where col_ptr[0] denotes the index offset)

Although not checked at runtime, row_indx are expected to be in the range:

- 0 <= row_indx[i] <= n-1 when ‘‘col_ptr[0]‘‘=0.
- 1 <= row_indx[i] <= n when ‘‘col_ptr[0]‘‘=1.

col_ptr Input parameter

col_ptr is const armpl_int_t*

col_ptr is an array that provides the index into row_indx and vals for each column of the sparse matrix A. i.e. row_indx[col_ptr[i]] is the first non-zero row index present in column i.

Constraints: Array of length n+1

col_ptr[0] must be 0 or 1 denoting array indexing from 0 (C) or 1 (Fortran).

col_ptr[n] will be the (total number of non-zeroes)+‘‘col_ptr[0]‘‘.

vals Input parameter

vals is const armpl_singlecomplex_t*

Non-zero entries in the matrix to be represented in the sparse matrix structure.

Constraint: Array of length NNZ

flags Input parameter

flags is armpl_int_t

Use this flag to create certain properties of the matrix. This can be set to:

- 0. No restrictions apply. The function will make a copy of the input arrays, which can be deallocated by the user on return from this function in order to minimize memory usage.
- ARMPL_SPARSE_CREATE_NOCOPY. The function will not make a copy of the input arrays. These input arrays, therefore, **must not** be deallocated until after a call to armpl_spmat_destroy with the armpl_spmat object returned by this function. This will reduce the scope of optimizations that might be applied in subsequent calls to armpl_spmv_optimize.

6.3.6 armpl_spmat_create_csc_d

To create a real double precision sparse matrix object from input data given in Compressed Sparse Column (CSC) format, use **armpl_spmat_create_csc_d**.

Syntax

C specification:

```
#include "armpl.h"

armpl_status_t armpl_spmat_create_csc_d(armpl_spmat_t *A, armpl_int_t m,
                                        armpl_int_t n,
                                        const armpl_int_t *row_indx,
                                        const armpl_int_t *col_ptr,
                                        const double *vals,
                                        armpl_int_t flags);
```

Fortran 2003 specification:

```
use armpl_library

subroutine armpl_spmat_create_csc(A, m, n, row_indx, col_ptr, vals, flags, info)
         use armpl_kinds
         integer(kind=armpl_i8), intent(out) :: A
         integer, intent(in) :: m, n
         integer, intent(in) :: row_indx(*)
         integer, intent(in) :: col_ptr(n+1)
         real(kind=armpl_r64), intent(in) :: vals(*)
         integer, intent(in) :: flags
         integer, intent(out) :: info
end subroutine armpl_spmat_create_csc
```

Returns

In C/C++, this function returns an `armpl_status_t`, which can be tested to check that the function returned correctly. In Fortran, this value is stored in the parameter `info`.

Possible return values are:

- **`ARMPL_STATUS_SUCCESS`** The function returned with no errors.
- **`ARMPL_STATUS_INPUT_PARAMETER_ERROR`** An error was found with the supplied input parameters.
- **`ARMPL_STATUS_EXECUTION_FAILURE`** The function failed for an alternative reason.

Parameters

`A` Output parameter

`A` is `armpl_spmat_t*`

The sparse matrix object to be created.

`m` Input parameter

`m` is `armpl_int_t`

`m` is the number of rows in the sparse matrix.

Constraint: `m` >= 0

`n` Input parameter

`n` is `armpl_int_t`

`n` is the number of columns in the sparse matrix.

Constraint: `n` >= 0

`row_indx` Input parameter

`row_indx` is `const armpl_int_t*`

`row_indx` is an array listing the rows for each non-zero entry.

Constraints: Array of length NNZ = `col_ptr[M] - col_ptr[0]` (where `col_ptr[0]` denotes the index offset).

Although not checked at runtime, `row_indx` are expected to be in the range:

- 0 <= `row_indx[i]` <= `n-1` when ‘‘col_ptr[0]‘‘=0.
- 1 <= `row_indx[i]` <= `n` when ‘‘col_ptr[0]‘‘=1.

`col_ptr` Input parameter

`col_ptr` is `const armpl_int_t*`

`col_ptr` is an array that provides the index into `row_indx` and `vals` for each column of the sparse matrix A. i.e. `row_indx[col_ptr[i]]` is the first non-zero row index present in column `i`.

Constraints: Array of length `n+1`

`col_ptr[0]` must be 0 or 1 denoting array indexing from 0 (C) or 1 (Fortran).

`col_ptr[n]` will be the (total number of non-zeroes)+‘‘col_ptr[0]‘‘.

`vals` Input parameter

`vals` is `const double*`

Non-zero entries in the matrix to be represented in the sparse matrix structure.

Constraint: Array of length NNZ

`flags` Input parameter

`flags` is `armpl_int_t`

Use this flag to create certain properties of the matrix. This can be set to:

- `0`. No restrictions apply. The function will make a copy of the input arrays, which can be deallocated by the user on return from this function in order to minimize memory usage.
- `ARMPL_SPARSE_CREATE_NOCOPY`. The function will not make a copy of the input arrays. These input arrays, therefore, **must not** be deallocated until after a call to `armpl_spmat_destroy` with the `armpl_spmat` object returned by this function. This will reduce the scope of optimizations that might be applied in subsequent calls to `armpl_spmv_optimize`.

6.3.7 armpl_spmat_create_csc_s

To create a real single precision sparse matrix object from input data given in Compressed Sparse Column (CSC) format, use **armpl_spmat_create_csc_s**.

Syntax

C specification:

```
#include "armpl.h"

armpl_status_t armpl_spmat_create_csc_s(armpl_spmat_t *A, armpl_int_t m,
                                        armpl_int_t n,
                                        const armpl_int_t *row_indx,
                                        const armpl_int_t *col_ptr,
                                        const float *vals,
                                        armpl_int_t flags);
```

Fortran 2003 specification:

```
use armpl_library

subroutine armpl_spmat_create_csc(A, m, n, row_indx, col_ptr, vals, flags, info)
         use armpl_kinds
         integer(kind=armpl_i8), intent(out) :: A
         integer, intent(in) :: m, n
         integer, intent(in) :: row_indx(*)
         integer, intent(in) :: col_ptr(n+1)
         real(kind=armpl_r32), intent(in) :: vals(*)
         integer, intent(in) :: flags
         integer, intent(out) :: info
end subroutine armpl_spmat_create_csc
```

Returns

In C/C++ this function returns an `armpl_status_t` which can be tested to check that the function returned correctly. In Fortran, this value is stored in the parameter `info`.

Possible return values are:

- **`ARMPL_STATUS_SUCCESS`** The function returned with no errors.
- **`ARMPL_STATUS_INPUT_PARAMETER_ERROR`** An error was found with the supplied input parameters.
- **`ARMPL_STATUS_EXECUTION_FAILURE`** The function failed for an alternative reason.

Parameters

`A` Output parameter

`A` is `armpl_spmat_t*`

The sparse matrix object to be created.

`m` Input parameter

`m` is `armpl_int_t`

`m` is the number of rows in the sparse matrix.

Constraint: `m` >= 0

`n` Input parameter

`n` is `armpl_int_t`

`n` is the number of columns in the sparse matrix.

Constraint: `n` >= 0

`row_indx` Input parameter

`row_indx` is `const armpl_int_t*`

`row_indx` is an array listing the rows for each non-zero entry.

Constraints: Array of length NNZ = `col_ptr[M] - col_ptr[0]` (where `col_ptr[0]` denotes the index offset)

Although not checked at runtime, `row_indx` are expected to be in the range:

- 0 <= `row_indx[i]` <= `n-1` when ‘‘col_ptr[0]‘‘=0.
- 1 <= `row_indx[i]` <= `n` when ‘‘col_ptr[0]‘‘=1.

col_ptr Input parameter

col_ptr is const armpl_int_t*

col_ptr is an array that provides the index into row_indx and vals for each column of the sparse matrix A. i.e. row_indx[col_ptr[i]] is the first non-zero row index present in column i.

Constraints: Array of length n+1

col_ptr[0] must be 0 or 1 denoting array indexing from 0 (C) or 1 (Fortran).

col_ptr[n] will be the (total number of non-zeroes)+‘‘col_ptr[0]‘‘.

vals Input parameter

vals is const float*

Non-zero entries in the matrix to be represented in the sparse matrix structure.

Constraint: Array of length NNZ

flags Input parameter

flags is armpl_int_t

Use this flag to create certain properties of the matrix. This can be set to:

- 0. No restrictions apply. The function will make a copy of the input arrays, which can be deallocated by the user on return from this function in order to minimize memory usage.
- ARMPL_SPARSE_CREATE_NOCOPY. The function will not make a copy of the input arrays. These input arrays, therefore, **must not** be deallocated until after a call to armpl_spmat_destroy with the armpl_spmat object returned by this function. This will reduce the scope of optimizations that might be applied in subsequent calls to armpl_spmv_optimize.

6.3.8 armpl_spmat_create_csc_z

To create a complex double precision sparse matrix object from input data given in Compressed Sparse Column (CSC) format, use **armpl_spmat_create_csc_z**.

Syntax

C specification:

```
#include "armpl.h"

armpl_status_t armpl_spmat_create_csc_z(armpl_spmat_t *A, armpl_int_t m,
                                        armpl_int_t n,
                                        const armpl_int_t *row_indx,
                                        const armpl_int_t *col_ptr,
                                        const armpl_doublecomplex_t *vals,
                                        armpl_int_t flags);
```

Fortran 2003 specification:

```
use armpl_library

subroutine armpl_spmat_create_csc(A, m, n, row_indx, col_ptr, vals, flags, info)
         use armpl_kinds
         integer(kind=armpl_i8), intent(out) :: A
         integer, intent(in) :: m, n
         integer, intent(in) :: row_indx(*)
         integer, intent(in) :: col_ptr(n+1)
         complex(kind=armpl_r64), intent(in) :: vals(*)
         integer, intent(in) :: flags
```

(continues on next page)

(continued from previous page)

```
        integer, intent(out) :: info
end subroutine armpl_spmat_create_csc
```

Returns

In C/C++, this function returns an `armpl_status_t`, which can be tested to check that the function returned correctly. In Fortran, this value is stored in the parameter `info`.

Possible return values are:

- **`ARMPL_STATUS_SUCCESS`** The function returned with no errors.
- **`ARMPL_STATUS_INPUT_PARAMETER_ERROR`** An error was found with the supplied input parameters.
- **`ARMPL_STATUS_EXECUTION_FAILURE`** The function failed for an alternative reason.

Parameters

`A` Output parameter

`A` is `armpl_spmat_t*`

The sparse matrix object to be created.

`m` Input parameter

`m` is `armpl_int_t`

`m` is the number of rows in the sparse matrix.

Constraint: `m` >= 0

`n` Input parameter

`n` is `armpl_int_t`

`n` is the number of columns in the sparse matrix.

Constraint: `n` >= 0

`row_indx` Input parameter

`row_indx` is `const armpl_int_t*`

`row_indx` is an array listing the rows for each non-zero entry.

Constraints: Array of length NNZ = `col_ptr[M] - col_ptr[0]` (where `col_ptr[0]` denotes the index offset).

Although not checked at runtime, `row_indx` are expected to be in the range:

- 0 <= `row_indx[i]` <= `n-1` when ‘‘col_ptr[0]‘‘=0.
- 1 <= `row_indx[i]` <= `n` when ‘‘col_ptr[0]‘‘=1.

`col_ptr` Input parameter

`col_ptr` is `const armpl_int_t*`

`col_ptr` is an array that provides the index into `row_indx` and `vals` for each column of the sparse matrix A. i.e. `row_indx[col_ptr[i]]` is the first non-zero row index present in column `i`.

Constraints: Array of length `n+1`

`col_ptr[0]` must be 0 or 1 denoting array indexing from 0 (C) or 1 (Fortran).

`col_ptr[n]` will be the (total number of non-zeroes)+‘‘col_ptr[0]‘‘.

vals Input parameter

vals is const armpl_doublecomplex_t*

Non-zero entries in the matrix to be represented in the sparse matrix structure.

Constraint: Array of length NNZ.

flags Input parameter

flags is armpl_int_t

Use this flag to create certain properties of the matrix. This can be set to:

- 0. No restrictions apply. The function will make a copy of the input arrays, which can be deallocated by the user on return from this function in order to minimize memory usage.
- ARMPL_SPARSE_CREATE_NOCOPY. The function will not make a copy of the input arrays. These input arrays, therefore, **must not** be deallocated until after a call to armpl_spmat_destroy with the armpl_spmat object returned by this function. This will reduce the scope of optimizations that might be applied in subsequent calls to armpl_spmv_optimize.

6.3.9 armpl_spmat_create_csr_c

To create a complex single precision sparse matrix object from input data given in Compressed Sparse Row (CSR) format, use **armpl_spmat_create_csr_c**.

Syntax

C specification:

```
#include "armpl.h"

armpl_status_t armpl_spmat_create_csr_c(armpl_spmat_t *A, armpl_int_t m,
                                        armpl_int_t n,
                                        const armpl_int_t *row_ptr,
                                        const armpl_int_t *col_indx,
                                        const armpl_singlecomplex_t *vals,
                                        armpl_int_t flags);
```

Fortran 2003 specification:

```
use armpl_library

subroutine armpl_spmat_create_csr(A, m, n, row_ptr, col_indx, vals, flags, info)
         use armpl_kinds
         integer(kind=armpl_i8), intent(out) :: A
         integer, intent(in) :: m, n
         integer, intent(in) :: row_ptr(m+1)
         integer, intent(in) :: col_indx(*)
         complex(kind=armpl_r32), intent(in) :: vals(*)
         integer, intent(in) :: flags
         integer, intent(out) :: info
end subroutine armpl_spmat_create_csr
```

Returns

In C/C++, this function returns an armpl_status_t, which can be tested to check that the function returned correctly. In Fortran, this value is stored in the parameter info.

Possible return values are:

- **ARMPL_STATUS_SUCCESS** The function returned with no errors.
- **ARMPL_STATUS_INPUT_PARAMETER_ERROR** An error was found with the supplied input parameters.
- **ARMPL_STATUS_EXECUTION_FAILURE** The function failed for an alternative reason.

Parameters

A Output parameter

A is armpl_spmat_t*

The sparse matrix object to be created.

m Input parameter

m is armpl_int_t

m is the number of rows in the sparse matrix.

Constraint: m >= 0

n Input parameter

n is armpl_int_t

n is the number of columns in the sparse matrix.

Constraint: n >= 0

row_ptr Input parameter

row_ptr is const armpl_int_t*

row_ptr is an array that provides the index into col_indx and vals for each row of the sparse matrix A. i.e. col_indx[row_ptr[i]] is the first non-zero column index present in row i.

Constraint: Array of length m+1

row_ptr[0] must be 0 or 1 denoting array indexing from 0 (C) or 1 (Fortran).

row_ptr[m] will be the (total number of non-zeroes)+‘‘row_ptr[0]‘‘

col_indx Input parameter

col_indx is const armpl_int_t*

col_indx is an array listing the columns for each non-zero entry.

Constraints Array of length NNZ = row_ptr[M] - row_ptr[0] (where row_ptr[0] denotes the index offset).

Although not checked at runtime, col_indx are expected to be in the range:

- 0 <= row_indx[i] <= n-1 when ‘‘col_ptr[0]‘‘=0.
- 1 <= row_indx[i] <= n when ‘‘col_ptr[0]‘‘=1.

vals Input parameter

vals is const armpl_singlecomplex_t*

Non-zero entries in the matrix to be represented in the sparse matrix structure.

Constraint: Array of length NNZ.

flags

Input parameter

flags is armpl_int_t

Use this flag to create certain properties of the matrix. This can be set to:

- `0`. No restrictions apply. The function will make a copy of the input arrays, which can be deallocated by the user on return from this function in order to minimize memory usage.
- `ARMPL_SPARSE_CREATE_NOCOPY`. The function will not make a copy of the input arrays. These input arrays, therefore, **must not** be deallocated until after a call to `armpl_spmat_destroy` with the `armpl_spmat` object returned by this function. This will reduce the scope of optimizations that might be applied in subsequent calls to `armpl_spmv_optimize`.

6.3.10 armpl_spmat_create_csr_d

To create a real double precision sparse matrix object from input data given in Compressed Sparse Row (CSR) format, **armpl_spmat_create_csr_d**.

Syntax

C specification:

```
#include "armpl.h"

armpl_status_t armpl_spmat_create_csr_d(armpl_spmat_t *A, armpl_int_t m,
                                        armpl_int_t n,
                                        const armpl_int_t *row_ptr,
                                        const armpl_int_t *col_indx,
                                        const double *vals,
                                        armpl_int_t flags);
```

Fortran 2003 specification:

```
use armpl_library

subroutine armpl_spmat_create_csr(A, m, n, row_ptr, col_indx, vals, flags, info)
         use armpl_kinds
         integer(kind=armpl_i8), intent(out) :: A
         integer, intent(in) :: m, n
         integer, intent(in) :: row_ptr(m+1)
         integer, intent(in) :: col_indx(*)
         real(kind=armpl_r64), intent(in) :: vals(*)
         integer, intent(in) :: flags
         integer, intent(out) :: info
end subroutine armpl_spmat_create_csr
```

Returns

In C/C++, this function returns an `armpl_status_t`, which can be tested to check that the function returned correctly. In Fortran, this value is stored in the parameter `info`.

Possible return values are:

- **`ARMPL_STATUS_SUCCESS`** The function returned with no errors.
- **`ARMPL_STATUS_INPUT_PARAMETER_ERROR`** An error was found with the supplied input parameters.
- **`ARMPL_STATUS_EXECUTION_FAILURE`** The function failed for an alternative reason.

Parameters

`A` Output parameter

`A` is `armpl_spmat_t*`

The sparse matrix object to be created.

m Input parameter

m is armpl_int_t

m is the number of rows in the sparse matrix.

Constraint: m >= 0

n Input parameter

n is armpl_int_t

n is the number of columns in the sparse matrix.

Constraint: n >= 0

row_ptr Input parameter

row_ptr is const armpl_int_t*

row_ptr is an array that provides the index into col_indx and vals for each row of the sparse matrix A. i.e. col_indx[row_ptr[i]] is the first non-zero column index present in row i.

Constraints: Array of length m+1

row_ptr[0] must be 0 or 1 denoting array indexing from 0 (C) or 1 (Fortran).

row_ptr[m] will be the (total number of non-zeroes)+‘‘row_ptr[0]‘‘.

col_indx Input parameter

col_indx is const armpl_int_t*

col_indx is an array listing the columns for each non-zero entry.

Constraints Array of length NNZ = row_ptr[M] - row_ptr[0] (where row_ptr[0] denotes the index offset).

Although not checked at runtime, col_indx are expected to be in the range:

- 0 <= row_indx[i] <= n-1 when ‘‘col_ptr[0]‘‘=0.
- 1 <= row_indx[i] <= n when ‘‘col_ptr[0]‘‘=1.

vals Input parameter

vals is const double*

Non-zero entries in the matrix to be represented in the sparse matrix structure.

Constraint: Array of length NNZ

flags Input parameter

flags is armpl_int_t

Use this flag to create certain properties of the matrix. This can be set to:

- 0. No restrictions apply. The function will make a copy of the input arrays, which can be deallocated by the user on return from this function in order to minimize memory usage.
- ARMPL_SPARSE_CREATE_NOCOPY. The function will not make a copy of the input arrays. These input arrays, therefore, **must not** be deallocated until after a call to armpl_spmat_destroy with the armpl_spmat object returned by this function. This will reduce the scope of optimizations that might be applied in subsequent calls to armpl_spmv_optimize.

6.3.11 armpl_spmat_create_csr_s

To create a real single precision sparse matrix object from input data given in Compressed Sparse Row (CSR) format, use **armpl_spmat_create_csr_s**.

Syntax

C specification:

```
#include "armpl.h"

armpl_status_t armpl_spmat_create_csr_s(armpl_spmat_t *A, armpl_int_t m,
                                        armpl_int_t n,
                                        const armpl_int_t *row_ptr,
                                        const armpl_int_t *col_indx,
                                        const float *vals,
                                        armpl_int_t flags);
```

Fortran 2003 specification:

```
use armpl_library

subroutine armpl_spmat_create_csr(A, m, n, row_ptr, col_indx, vals, flags, info)
         use armpl_kinds
         integer(kind=armpl_i8), intent(out) :: A
         integer, intent(in) :: m, n
         integer, intent(in) :: row_ptr(m+1)
         integer, intent(in) :: col_indx(*)
         real(kind=armpl_r32), intent(in) :: vals(*)
         integer, intent(in) :: flags
         integer, intent(out) :: info
end subroutine armpl_spmat_create_csr
```

Returns

In C/C++, this function returns an `armpl_status_t`, which can be tested to check that the function returned correctly. In Fortran, this value is stored in the parameter `info`.

Possible return values are:

- **`ARMPL_STATUS_SUCCESS`** The function returned with no errors.
- **`ARMPL_STATUS_INPUT_PARAMETER_ERROR`** An error was found with the supplied input parameters.
- **`ARMPL_STATUS_EXECUTION_FAILURE`** The function failed for an alternative reason.

Parameters

`A` Output parameter

`A` is `armpl_spmat_t*`

The sparse matrix object to be created.

`m` Input parameter

`m` is `armpl_int_t`

`m` is the number of rows in the sparse matrix.

Constraint: `m` >= 0

`n` Input parameter

`n` is `armpl_int_t`

`n` is the number of columns in the sparse matrix.

Constraint: `n` >= 0

row_ptr Input parameter

row_ptr is const armpl_int_t*

row_ptr is an array that provides the index into col_indx and vals for each row of the sparse matrix A. i.e. col_indx[row_ptr[i]] is the first non-zero column index present in row i.

Constraint: Array of length m+1

row_ptr[0] must be 0 or 1 denoting array indexing from 0 (C) or 1 (Fortran).

row_ptr[m] will be the (total number of non-zeroes)+‘‘row_ptr[0]‘‘.

col_indx Input parameter

col_indx is const armpl_int_t*

col_indx is an array listing the columns for each non-zero entry.

Constraints Array of length NNZ = row_ptr[M] - row_ptr[0] (where row_ptr[0] denotes the index offset).

Although not checked at runtime, col_indx are expected to be in the range:

- 0 <= row_indx[i] <= n-1 when ‘‘col_ptr[0]‘‘=0.
- 1 <= row_indx[i] <= n when ‘‘col_ptr[0]‘‘=1.

vals Input parameter

vals is const float*

Non-zero entries in the matrix to be represented in the sparse matrix structure.

Constraint: Array of length NNZ.

flags Input parameter

flags is armpl_int_t

Use this flag to create certain properties of the matrix. This can be set to:

- 0. No restrictions apply. The function will make a copy of the input arrays, which can be deallocated by the user on return from this function in order to minimize memory usage.
- ARMPL_SPARSE_CREATE_NOCOPY. The function will not make a copy of the input arrays. These input arrays, therefore, **must not** be deallocated until after a call to armpl_spmat_destroy with the armpl_spmat object returned by this function. This will reduce the scope of optimizations that might be applied in subsequent calls to armpl_spmv_optimize.

6.3.12 armpl_spmat_create_csr_z

To create a complex double precision sparse matrix object from input data given in Compressed Sparse Row (CSR) format, use **armpl_spmat_create_csr_z**.

Syntax

C specification:

```
#include "armpl.h"

armpl_status_t armpl_spmat_create_csr_z(armpl_spmat_t *A, armpl_int_t m,
                                        armpl_int_t n,
                                        const armpl_int_t *row_ptr,
                                        const armpl_int_t *col_indx,
                                        const armpl_doublecomplex_t *vals,
                                        armpl_int_t flags);
```

Fortran 2003 specification:

```
use armpl_library

subroutine armpl_spmat_create_csr(A, m, n, row_ptr, col_indx, vals, flags, info)
         use armpl_kinds
         integer(kind=armpl_i8), intent(out) :: A
         integer, intent(in) :: m, n
         integer, intent(in) :: row_ptr(m+1)
         integer, intent(in) :: col_indx(*)
         complex(kind=armpl_r64), intent(in) :: vals(*)
         integer, intent(in) :: flags
         integer, intent(out) :: info
end subroutine armpl_spmat_create_csr
```

Returns

In C/C++, this function returns an `armpl_status_t`, which can be tested to check that the function returned correctly. In Fortran, this value is stored in the parameter `info`.

Possible return values are:

- **`ARMPL_STATUS_SUCCESS`** The function returned with no errors.
- **`ARMPL_STATUS_INPUT_PARAMETER_ERROR`** An error was found with the supplied input parameters.
- **`ARMPL_STATUS_EXECUTION_FAILURE`** The function failed for an alternative reason.

Parameters

`A` Output parameter

`A` is `armpl_spmat_t*`

The sparse matrix object to be created.

`m` Input parameter

`m` is `armpl_int_t`

`m` is the number of rows in the sparse matrix.

Constraint: `m` >= 0

`n` Input parameter

`n` is `armpl_int_t`

`n` is the number of columns in the sparse matrix.

Constraint: `n` >= 0

`row_ptr` Input parameter

`row_ptr` is `const armpl_int_t*`

`row_ptr` is an array that provides the index into `col_indx` and `vals` for each row of the sparse matrix A. i.e. `col_indx[row_ptr[i]]` is the first non-zero column index present in row `i`.

Constraint: Array of length `m+1`

`row_ptr[0]` must be 0 or 1 denoting array indexing from 0 (C) or 1 (Fortran).

`row_ptr[m]` will be the (total number of non-zeroes)+‘‘row_ptr[0]‘‘.

col_indx Input parameter

col_indx is const armpl_int_t*

col_indx is an array listing the columns for each non-zero entry.

Constraints Array of length NNZ = row_ptr[M] - row_ptr[0] (where row_ptr[0] denotes the index offset).

Although not checked at runtime, col_indx are expected to be in the range:

- 0 <= row_indx[i] <= n-1 when ‘‘col_ptr[0]‘‘=0.
- 1 <= row_indx[i] <= n when ‘‘col_ptr[0]‘‘=1.

vals Input parameter

vals is const armpl_doublecomplex_t*

Non-zero entries in the matrix to be represented in the sparse matrix structure.

Constraint: Array of length NNZ

flags Input parameter

flags is armpl_int_t

Use this flag to create certain properties of the matrix. This can be set to:

- 0. No restrictions apply. The function will make a copy of the input arrays, which can be deallocated by the user on return from this function in order to minimize memory usage.
- ARMPL_SPARSE_CREATE_NOCOPY. The function will not make a copy of the input arrays. These input arrays, therefore, **must not** be deallocated until after a call to armpl_spmat_destroy with the armpl_spmat object returned by this function. This will reduce the scope of optimizations that might be applied in subsequent calls to armpl_spmv_optimize.

6.3.13 armpl_spmat_update_c

armpl_spmat_update_c is a utility function to update the values of a previously created sparse matrix object.

The number of entries to be updated is specified such that:

```
For i = 0, n_updates-1

     I = row_index[i]

     J = col_indx[i]

     a_(I,J) = vals[i]
```

Note that this routine cannot change the sparsity pattern, only update the values. A new armpl_spmat_t object should be created if a different structure is needed.

Syntax

C specification:

```
#include "armpl.h"


armpl_status_t armpl_spmat_update_c(armpl_spmat_t A, armpl_int_t n_updates,
                                    const armpl_int_t *row_indx,
                                    const armpl_int_t *col_indx,
                                    const armpl_singlecomplex_t *vals);
```

Fortran 2003 specification:

```
use armpl_library

subroutine armpl_spmat_update(A, n_updates, row_indx, col_indx, vals, info)
         use armpl_kinds
         integer(kind=armpl_i8), intent(in) :: A
         integer, intent(in) :: n_updates
         integer, intent(in) :: row_indx(n_updates)
         integer, intent(in) :: col_indx(n_updates)
         complex(kind=armpl_r32), intent(in) :: vals(n_updates)
         integer, intent(out) :: info
end subroutine armpl_spmat_update
```

Returns

In C/C++ this function returns an `armpl_status_t` which can be tested to check that the function returned correctly. In Fortran, this value is stored in the parameter `info`.

Possible return values are:

- **`ARMPL_STATUS_SUCCESS`** The function returned with no errors.
- **`ARMPL_STATUS_INPUT_PARAMETER_ERROR`** An error was found with the supplied input parameters.
- **`ARMPL_STATUS_EXECUTION_FAILURE`** The function failed for an alternative reason.

Parameters

`A` Input and output parameter

`A` is `armpl_spmat`

`A` is a sparse matrix structure.

Constraint: The sparse matrix `A` should have been previously created using an appropriate `armpl_spmat_create_*()` function.

`n_updates` Input parameter

`n_updates` is `armpl_int_t`

`n_updates` gives the number of values to be updated. It is therefore the length of the arrays `row_indx`, `col_indx` and `vals`.

Constraint: `n_updates >= 0`. `ARMPL_STATUS_SUCCESS` is returned if `n_updates=0`.

`row_indx` Input parameter

`row_indx` is `const armpl_int_t*`

`row_indx` is a pointer to an array containing the row indices of the values to be updated.

Constraint: The length of the array is `n_updates`.

`col_indx` Input parameter

`col_indx` is `const armpl_int_t*`

`col_indx` is a pointer to an array containing the column indices of the values to be updated.

Constraint: The length of the array is `n_updates`.

`vals` Input parameter

`vals` is `const armpl_singlecomplex_t*`

`vals` is a pointer to an array with the new non-zero entries to be entered.

Constraint: The length of the array is `n_updates`.

6.3.14 armpl_spmat_update_d

armpl_spmat_update_d is a utility function to update the values of a previously created sparse matrix object.

The number of entries to be updated is specified such that:

```
For i = 0, n_updates-1

     I = row_index[i]

     J = col_indx[i]

     a_(I,J) = vals[i]
```

Note: This routine cannot change the sparsity pattern, only update the values. A new `armpl_spmat_t` object should be created if a different structure is needed.

Syntax

C specification:

```
#include "armpl.h"


armpl_status_t armpl_spmat_update_d(armpl_spmat_t A, armpl_int_t n_updates,
                                    const armpl_int_t *row_indx,
                                    const armpl_int_t *col_indx,
                                    const double *vals);
```

Fortran 2003 specification:

```
use armpl_library

subroutine armpl_spmat_update(A, n_updates, row_indx, col_indx, vals, info)
         use armpl_kinds
         implicit none
         integer(kind=armpl_i8), intent(in) :: A
         integer, intent(in) :: n_updates
         integer, intent(in) :: row_indx(n_updates)
         integer, intent(in) :: col_indx(n_updates)
         real(kind=armpl_r64), intent(in) :: vals(n_updates)
         integer, intent(out) :: info
end subroutine armpl_spmat_update
```

Returns

In C/C++, this function returns an `armpl_status_t`, which can be tested to check that the function returned correctly. In Fortran, this value is stored in the parameter `info`.

Possible return values are:

- **ARMPL_STATUS_SUCCESS** The function returned with no errors.
- **ARMPL_STATUS_INPUT_PARAMETER_ERROR** An error was found with the supplied input parameters.
- **ARMPL_STATUS_EXECUTION_FAILURE** The function failed for an alternative reason.

Parameters

A Input and output parameter

A is `armpl_spmat`

A is a sparse matrix structure.

Constraint: The sparse matrix `A` should have been previously created using an appropriate `armpl_spmat_create_*()` function.

n_updates Input parameter

`n_updates` is `armpl_int_t`

`n_updates` gives the number of values to be updated. It is therefore the length of the arrays `row_indx`, `col_indx` and `vals`.

Constraint: `n_updates >= 0`. `ARMPL_STATUS_SUCCESS` is returned if `n_updates=0`.

row_indx Input parameter

`row_indx` is `const armpl_int_t*`

`row_indx` is a pointer to an array containing the row indices of the values to be updated.

Constraint: The length of the array is `n_updates`.

col_indx Input parameter

`col_indx` is `const armpl_int_t*`

`col_indx` is a pointer to an array containing the column indices of the values to be updated.

Constraint: The length of the array is `n_updates`.

vals Input parameter

`vals` is `const double*`

`vals` is a pointer to an array with the new non-zero entries to be entered.

Constraint: The length of the array is `n_updates`.

6.3.15 armpl_spmat_update_s

armpl_spmat_update_s is a utility function to update the values of a previously created sparse matrix object.

The number of entries to be updated is specified such that:

```
For i = 0, n_updates-1

     I = row_index[i]

     J = col_indx[i]

     a_(I,J) = vals[i]
```

Note: This routine cannot change the sparsity pattern, only update the values. A new `armpl_spmat_t` object should be created if a different structure is needed.

Syntax

C specification:

```
#include "armpl.h"

armpl_status_t armpl_spmat_update_s(armpl_spmat_t A, armpl_int_t n_updates,
                                    const armpl_int_t *row_indx,
                                    const armpl_int_t *col_indx,
                                    const float *vals);
```

Fortran 2003 specification:

```
use armpl_library

subroutine armpl_spmat_update(A, n_updates, row_indx, col_indx, vals, info)
         use armpl_kinds
         integer(kind=armpl_i8), intent(in) :: A
         integer, intent(in) :: n_updates
         integer, intent(in) :: row_indx(n_updates)
         integer, intent(in) :: col_indx(n_updates)
         real(kind=armpl_r32), intent(in) :: vals(n_updates)
         integer, intent(out) :: info
end subroutine armpl_spmat_update
```

Returns

In C/C++ this function returns an `armpl_status_t` which can be tested to check that the function returned correctly. In Fortran, this value is stored in the parameter `info`.

Possible return values are:

- **`ARMPL_STATUS_SUCCESS`** The function returned with no errors.
- **`ARMPL_STATUS_INPUT_PARAMETER_ERROR`** An error was found with the supplied input parameters.
- **`ARMPL_STATUS_EXECUTION_FAILURE`** The function failed for an alternative reason.

Parameters

`A` Input and output parameter

`A` is `armpl_spmat`

`A` is a sparse matrix structure.

Constraint: The sparse matrix `A` should have been previously created using an appropriate `armpl_spmat_create_*()` function.

`n_updates` Input parameter

`n_updates` is `armpl_int_t`

`n_updates` gives the number of values to be updated. It is therefore the length of the arrays `row_indx`, `col_indx` and `vals`.

Constraint: `n_updates >= 0`. `ARMPL_STATUS_SUCCESS` is returned if `n_updates=0`.

`row_indx` Input parameter

`row_indx` is `const armpl_int_t*`

`row_indx` is a pointer to an array containing the row indices of the values to be updated.

Constraint: The length of the array is `n_updates`.

`col_indx` Input parameter

`col_indx` is `const armpl_int_t*`

`col_indx` is a pointer to an array containing the column indices of the values to be updated.

Constraint: The length of the array is `n_updates`.

`vals` Input parameter

`vals` is `const float*`

`vals` is a pointer to an array with the new non-zero entries to be entered.

Constraint: The length of the array is `n_updates`.

6.3.16 armpl_spmat_update_z

armpl_spmat_update_z is a utility function to update the values of a previously created sparse matrix object.

The number of entries to be updated is specified such that:

```
For i = 0, n_updates-1

     I = row_index[i]

     J = col_indx[i]

     a_(I,J) = vals[i]
```

Note: This routine cannot change the sparsity pattern, only update the values. A new `armpl_spmat_t` object should be created if a different structure is needed.

Syntax

C specification:

```
#include "armpl.h"


armpl_status_t armpl_spmat_update_z(armpl_spmat_t A, armpl_int_t n_updates,
                                    const armpl_int_t *row_indx,
                                    const armpl_int_t *col_indx,
                                    const armpl_doublecomplex_t *vals);
```

Fortran 2003 specification:

```
use armpl_library

subroutine armpl_spmat_update(A, n_updates, row_indx, col_indx, vals, info)
         use armpl_kinds
         integer(kind=armpl_i8), intent(in) :: A
         integer, intent(in) :: n_updates
         integer, intent(in) :: row_indx(n_updates)
         integer, intent(in) :: col_indx(n_updates)
         complex(kind=armpl_r64), intent(in) :: vals(n_updates)
         integer, intent(out) :: info
end subroutine armpl_spmat_update
```

Returns

In C/C++, this function returns an `armpl_status_t`, which can be tested to check that the function returned correctly. In Fortran, this value is stored in the parameter `info`.

Possible return values are:

- **ARMPL_STATUS_SUCCESS** The function returned with no errors.
- **ARMPL_STATUS_INPUT_PARAMETER_ERROR** An error was found with the supplied input parameters.
- **ARMPL_STATUS_EXECUTION_FAILURE** The function failed for an alternative reason.

Parameters

A Input and output parameter

A is armpl_spmat

A is a sparse matrix structure.

Constraint: The sparse matrix A should have been previously created using an appropriate armpl_spmat_create_*() function.

n_updates Input parameter

n_updates is armpl_int_t

n_updates gives the number of values to be updated. It is therefore the length of the arrays row_indx, col_indx and vals.

Constraint: n_updates >= 0. ARMPL_STATUS_SUCCESS is returned if n_updates=0.

row_indx Input parameter

row_indx is const armpl_int_t*

row_indx is a pointer to an array containing the row indices of the values to be updated.

Constraint: The length of the array is n_updates.

col_indx Input parameter

col_indx is const armpl_int_t*

col_indx is a pointer to an array containing the column indices of the values to be updated.

Constraint: The length of the array is n_updates.

vals Input parameter

vals is const armpl_doublecomplex_t*

vals is a pointer to an array with the new non-zero entries to be entered.

Constraint: The length of the array is n_updates.

6.3.17 armpl_spmat_hint

armpl_spmat_hint is a utility function to provide various hints that can be used during the optimization phase to allow the library to choose the best implementation.

Multiple hints can be given in separate calls, building extra information into the sparse matrix object each time.

Syntax

C specification:

```
#include "armpl.h"

armpl_status_t armpl_spmat_hint(armpl_spmat_t A, enum sparse_hint_type hint, enum␣
↪sparse_hint_value value);
```

Fortran 2003 specification:

```
use armpl_library

subroutine armpl_spmat_hint(A, sparse_hint_type, sparse_hint_value, info)
         use armpl_kinds
         integer(kind=armpl_i8), intent(in) :: A
         integer(kind=armpl_i4), intent(in) :: sparse_hint_type, sparse_hint_value
         integer, intent(out) :: info
end subroutine armpl_spmat_hint
```

Returns

In C/C++ this function returns an `armpl_status_t` which can be tested to check that the function returned correctly. In Fortran, this value is stored in the parameter `info`.

Possible return values are:

- **`ARMPL_STATUS_SUCCESS`** The function returned with no errors.
- **`ARMPL_STATUS_INPUT_PARAMETER_ERROR`** An error was found with the supplied input parameters.
- **`ARMPL_STATUS_EXECUTION_FAILURE`** The function failed for an alternative reason.

Parameters

`A` Input and output parameter

`A` is `armpl_spmat`

`A` is a sparse matrix structure.

Constraint: The sparse matrix `A` should have been previously created using an appropriate `armpl_spmat_create_*()` function

`hint` Input parameter

`hint` is `enum sparse_hint_type`

`hint` denotes the type of hint to be supplied.

Constraint: The `hint` should be either `ARMPL_SPARSE_HINT_STRUCTURE`, `ARMPL_SPARSE_HINT_SPMV_OPERATION`, `ARMPL_SPARSE_HINT_MEMORY`, `ARMPL_SPARSE_HINT_SPMV_INVOCATIONS`

`value` Input parameter

`value` is `enum sparse_hint_value`

`value` is the setting for the hint of type `hint`.

Constraint: The hints given should match the values shown in the table below.

Hint structure

Listed below are the various hints that are available for each `ARMPL_SPARSE_HINT_TYPE` option, where (*) indicates the default option for each type:

- `ARMPL_SPARSE_HINT_TYPE``==``ARMPL_SPARSE_HINT_MEMORY`

  `ARMPL_SPARSE_MEMORY_NOALLOCS`

  `ARMPL_SPARSE_MEMORY_ALLOCS` (*)

- `ARMPL_SPARSE_HINT_TYPE``==``ARMPL_SPARSE_HINT_STRUCTURE`

ARMPL_SPARSE_STRUCTURE_DENSE

ARMPL_SPARSE_STRUCTURE_UNSTRUCTURED (*)

ARMPL_SPARSE_STRUCTURE_SYMMETRIC

ARMPL_SPARSE_STRUCTURE_DIAGONAL

ARMPL_SPARSE_STRUCTURE_BLOCKDIAGONAL

ARMPL_SPARSE_STRUCTURE_BANDED

ARMPL_SPARSE_STRUCTURE_TRIANGULAR

ARMPL_SPARSE_STRUCTURE_BLOCKTRIANGULAR

ARMPL_SPARSE_STRUCTURE_HERMITIAN

ARMPL_SPARSE_STRUCTURE_HPCG

- ARMPL_SPARSE_HINT_TYPE``==``ARMPL_SPARSE_HINT_SPMV_INVOCATIONS

  ARMPL_SPARSE_INVOCATIONS_SINGLE

  ARMPL_SPARSE_INVOCATIONS_FEW

  ARMPL_SPARSE_INVOCATIONS_MANY (*)

- ARMPL_SPARSE_HINT_TYPE``==``ARMPL_SPARSE_HINT_SPMV_OPERATION

  ARMPL_SPARSE_OPERATION_NOTRANS (*)

  ARMPL_SPARSE_OPERATION_TRANS

  ARMPL_SPARSE_OPERATION_CONJTRANS

Note: If the armpl_spmat object was created using the ARMPL_SPARSE_CREATE_NOCOPY flag then that specfication takes precedence over the ARMPL_SPARSE_MEMORY_ALLOCS hint.

6.3.18 armpl_spmv_optimize

armpl_spmv_optimize performs optimizations on a previously created Arm Performance Libraries sparse matrix object appropriate to an SpMV operation.

This will use any previously supplied hints coming from *armpl_spmat_hint*.

This function does not perform the SpMV itself and will potentially create new data-structures inside the sparse matrix object.

These will contain alternative representations of the supplied sparse matrix in order to later calculate the SpMV operation more efficiently.

Syntax

C specification:

```
#include "armpl.h"

armpl_status_t armpl_spmv_optimize(armpl_spmat_t A);
```

Fortran 2003 specification:

```
use armpl_library

subroutine armpl_spmv_optimize(A, info)
         use armpl_kinds
         integer(kind=armpl_i8), intent(in) :: A
         integer, intent(out) :: info
end subroutine armpl_spmv_optimize
```

Returns

In C/C++, this function returns an `armpl_status_t`, which can be tested to check that the function returned correctly. In Fortran, this value is stored in the parameter `info`.

Possible return values are:

- **`ARMPL_STATUS_SUCCESS`** The function returned with no errors.
- **`ARMPL_STATUS_INPUT_PARAMETER_ERROR`** An error was found with the supplied input parameters.
- **`ARMPL_STATUS_EXECUTION_FAILURE`** The function failed for an alternative reason.

Parameters

`A` Input and output parameter

`A` is `armpl_spmat_t`

`A` is a sparse matrix structure.

Constraint: The sparse matrix `A` should have been previously created using an appropriate `armpl_spmat_create_*()` function.

6.3.19 armpl_spmv_exec_c

armpl_spmv_exec_c performs one of the sparse matrix-vector operations

```
y := alpha*A*x + beta*y,   or   y := alpha*A**T*x + beta*y,
```

where alpha and beta are scalars, x and y are vectors and A is a sparse matrix.

The sparse matrix, A, should have previously been created using a routine such as **armpl_spmat_create_csr_c**. It is not necessary to supply hints, using **armpl_spmv_hint()**, although it is recommended as the default settings might not be appropriate in every use-case. The optimize phase, **armpl_spmv_optimize()**, might be applied on the first execution of **armpl_spmv_exec_c** if it has not already been explicitly called by the user.

Syntax

C specification:

```
#include "armpl.h"

armpl_status_t armpl_spmv_exec_c(enum sparse_hint_value trans,
                                 armpl_singlecomplex_t alpha,
                                 armpl_spmat_t A,
                                 const armpl_singlecomplex_t *x,
                                 armpl_singlecomplex_t beta,
                                 armpl_singlecomplex_t *y);
```

Fortran 2003 specification:

```
use armpl_library

subroutine armpl_spmv_exec(trans, alpha, A, x, beta, y, info)
         use armpl_kinds
         integer(kind=armpl_i4), intent(in) :: trans
         complex(kind=armpl_r32), intent(in) :: alpha
         integer(kind=armpl_i8), intent(in) :: A
         complex(kind=armpl_r32), intent(in) :: x(*)
         complex(kind=armpl_r32), intent(in) :: beta
         complex(kind=armpl_r32), intent(inout) :: y(*)
         integer, intent(out) :: info
end subroutine armpl_spmv_exec
```

Returns

In C/C++ this function returns an `armpl_status_t` which can be tested to check that the function returned correctly. In Fortran, this value is stored in the parameter `info`.

Possible return values are:

- **`ARMPL_STATUS_SUCCESS`** The function returned with no errors.
- **`ARMPL_STATUS_INPUT_PARAMETER_ERROR`** An error was found with the supplied input parameters.
- **`ARMPL_STATUS_EXECUTION_FAILURE`** The function failed for an alternative reason.

Parameters

`trans`

Input parameter

`trans` is `enum sparse_hint_value`

`trans` describes the transpose operation to be performed on the sparse matrix A before calculating the matrix-vector multiplication.

Constraint: `trans` is either `ARMPL_SPARSE_OPERATION_NOTRANS`, `ARMPL_SPARSE_OPERATION_TRANS` or `ARMPL_SPARSE_OPERATION_CONJTRANS` for no transpose, transpose or performing a conjugate transpose, respectively.

`alpha` Input parameter

`alpha` is `armpl_singlecomplex_t`

`alpha` specifies the constant which premultiples the matrix A.

`A` Input and output parameter

`A` is `armpl_spmat`

`A` is a sparse matrix structure.

Constraint: The sparse matrix `A` should have been previously created using an appropriate `armpl_spmat_create_*_c()` function.

`x` Input parameter

`x` is `armpl_singlecomplex_t*`

`x` is the (dense) vector to be multiplied by the matrix A.

Constraint: `x` is a previously allocated array. This has length determined by the transpose type specified in `trans`. This length is given in terms of the dimensions of the sparse matrix object created in a `armpl_spmat_create_*_c` function: namely of length `n` when `trans=ARMPL_SPARSE_OPERATION_NOTRANS` and length `m` otherwise.

beta Input parameter

beta is armpl_singlecomplex_t

beta specifies the scalar to multiply each entry of the vector, y

Constraint: If beta = 0 then the array y is output only.

y Input and output parameter

y is armpl_singlecomplex_t

y gives the output of the sparse matrix-vector multiplication.

Constraint: y is a previously allocated array. This has length determined by the transpose type specified in trans. This length is given in terms of the dimensions of the sparse matrix object created in a armpl_spmat_create_*_c function: namely of length m when trans=ARMPL_SPARSE_OPERATION_NOTRANS and length n otherwise.

6.3.20 armpl_spmv_exec_d

armpl_spmv_exec_d performs one of the sparse matrix-vector operations

```
y := alpha*A*x + beta*y,   or   y := alpha*A**T*x + beta*y,
```

where alpha and beta are scalars, x and y are vectors and A is a sparse matrix.

The sparse matrix, A, should have previously been created using a routine such as **armpl_spmat_create_csr_d**. It is not necessary to supply hints, using **armpl_spmv_hint()**, although it is recommended as the default settings might not be appropriate in every use-case. The optimize phase, **armpl_spmv_optimize()**, might be applied on the first execution of **armpl_spmv_exec_d** if it has not already been explicitly called by the user.

Syntax

C specification:

```
#include "armpl.h"

armpl_status_t armpl_spmv_exec_d(enum sparse_hint_value trans, double alpha,
                                 armpl_spmat_t A, const double *x,
                                 double beta, double *y);
```

Fortran 2003 specification:

```
use armpl_library

subroutine armpl_spmv_exec(trans, alpha, A, x, beta, y, info)
         use armpl_kinds
         integer(kind=armpl_i4), intent(in) :: trans
         real(kind=armpl_r64), intent(in) :: alpha
               integer(kind=armpl_i8), intent(in) :: A
         real(kind=armpl_r64), intent(in) :: x(*)
         real(kind=armpl_r64), intent(in) :: beta
         real(kind=armpl_r64), intent(inout) :: y(*)
         integer, intent(out) :: info
end subroutine armpl_spmv_exec
```

Returns

In C/C++, this function returns an armpl_status_t, which can be tested to check that the function returned correctly. In Fortran, this value is stored in the parameter info.

Possible return values are:

- **ARMPL_STATUS_SUCCESS** The function returned with no errors.
- **ARMPL_STATUS_INPUT_PARAMETER_ERROR** An error was found with the supplied input parameters.
- **ARMPL_STATUS_EXECUTION_FAILURE** The function failed for an alternative reason.

Parameters

`trans`

Input parameter

`trans` is `enum sparse_hint_value`

`trans` describes the transpose operation to be performed on the sparse matrix A before calculating the matrix-vector multiplication.

Constraint: `trans` is either `ARMPL_SPARSE_OPERATION_NOTRANS`, `ARMPL_SPARSE_OPERATION_TRANS` or `ARMPL_SPARSE_OPERATION_CONJTRANS` for no transpose, transpose or performing a conjugate transpose, respectively.

alpha Input parameter

`alpha` is `double`

`alpha` specifies the constant which premultiples the matrix A.

A Input and output parameter

`A` is `armpl_spmat`

`A` is a sparse matrix structure.

Constraint: The sparse matrix `A` should have been previously created using an appropriate `armpl_spmat_create_*_d()` function.

x Input parameter

`x` is `double*`

`x` is the (dense) vector to be multiplied by the matrix A.

Constraint: `x` is a previously allocated array. This has length determined by the transpose type specified in `trans`. This length is given in terms of the dimensions of the sparse matrix object created in a `armpl_spmat_create_*_d` function: namely of length `n` when `trans=ARMPL_SPARSE_OPERATION_NOTRANS` and length `m` otherwise.

beta Input parameter

`beta` is `double`

`beta` specifies the scalar to multiply each entry of the vector, `y`.

Constraint: If `beta` = 0 then the array `y` is output only.

y Input and output parameter

`y` is `double`

`y` gives the output of the sparse matrix-vector multiplication.

Constraint: `y` is a previously allocated array. This has length determined by the transpose type specified in `trans`. This length is given in terms of the dimensions of the sparse matrix object created in a `armpl_spmat_create_*_d` function: namely of length `m` when `trans=ARMPL_SPARSE_OPERATION_NOTRANS` and length `n` otherwise.

6.3.21 armpl_spmv_exec_s

armpl_spmv_exec_s performs one of the sparse matrix-vector operations

```
y := alpha*A*x + beta*y,   or   y := alpha*A**T*x + beta*y,
```

where alpha and beta are scalars, x and y are vectors and A is a sparse matrix.

The sparse matrix, A, should have previously been created using a routine such as **armpl_spmat_create_csr_s**. It is not necessary to supply hints, using **armpl_spmv_hint**(), although it is recommended as the default settings might not be appropriate in every use-case. The optimize phase, **armpl_spmv_optimize**(), might be applied on the first execution of **armpl_spmv_exec_s** if it has not already been explicitly called by the user.

Syntax

C specification:

```
#include "armpl.h"

armpl_status_t armpl_spmv_exec_s(enum sparse_hint_value trans, float alpha,
                                 armpl_spmat_t A, const float *x, float beta,
                                 float *y);
```

Fortran 2003 specification:

```
use armpl_library

subroutine armpl_spmv_exec(trans, alpha, A, x, beta, y, info)
         use armpl_kinds
         integer(kind=armpl_i4), intent(in) :: trans
         real(kind=armpl_r32), intent(in) :: alpha
         integer(kind=armpl_i8), intent(in) :: A
         real(kind=armpl_r32), intent(in) :: x(*)
               real(kind=armpl_r32), intent(in) :: beta
         real(kind=armpl_r32), intent(inout) :: y(*)
         integer, intent(out) :: info
end subroutine armpl_spmv_exec
```

Returns

In C/C++, this function returns an `armpl_status_t`, which can be tested to check that the function returned correctly. In Fortran, this value is stored in the parameter `info`.

Possible return values are:

- **`ARMPL_STATUS_SUCCESS`** The function returned with no errors.
- **`ARMPL_STATUS_INPUT_PARAMETER_ERROR`** An error was found with the supplied input parameters.
- **`ARMPL_STATUS_EXECUTION_FAILURE`** The function failed for an alternative reason.

Parameters

`trans` Input parameter

`trans` is `enum sparse_hint_value`

`trans` describes the transpose operation to be performed on the sparse matrix A before calculating the matrix-vector multiplication.

Constraint: `trans` is either `ARMPL_SPARSE_OPERATION_NOTRANS`, `ARMPL_SPARSE_OPERATION_TRANS` or `ARMPL_SPARSE_OPERATION_CONJTRANS` for no transpose, transpose or performing a conjugate transpose, respectively.

`alpha` Input parameter

`alpha` is `float`

`alpha` specifies the constant which premultiples the matrix A.

`A` Input and output parameter

`A` is `armpl_spmat`

`A` is a sparse matrix structure.

Constraint: The sparse matrix `A` should have been previously created using an appropriate `armpl_spmat_create_*_s()` function.

`x`

Input parameter

`x` is `float*`

`x` is the (dense) vector to be multiplied by the matrix A.

Constraint: `x` is a previously allocated array. This has length determined by the transpose type specified in `trans`. This length is given in terms of the dimensions of the sparse matrix object created in a `armpl_spmat_create_*_s` function: namely of length `n` when `trans=ARMPL_SPARSE_OPERATION_NOTRANS` and length `m` otherwise.

`beta` Input parameter

`beta` is `float`

`beta` specifies the scalar to multiply each entry of the vector, `y`.

Constraint: If `beta` = 0 then the array `y` is output only.

`y` Input and output parameter

`y` is `float`

`y` gives the output of the sparse matrix-vector multiplication.

Constraint: `y` is a previously allocated array. This has length determined by the transpose type specified in `trans`. This length is given in terms of the dimensions of the sparse matrix object created in a `armpl_spmat_create_*_s` function: namely of length `m` when `trans=ARMPL_SPARSE_OPERATION_NOTRANS` and length `n` otherwise.

6.3.22 armpl_spmv_exec_z

armpl_spmv_exec_z performs one of the sparse matrix-vector operations

```
y := alpha*A*x + beta*y,   or   y := alpha*A**T*x + beta*y,
```

where alpha and beta are scalars, x and y are vectors and A is a sparse matrix.

The sparse matrix, A, should have previously been created using a routine such as **armpl_spmat_create_csr_z**. It is not necessary to supply hints, using **armpl_spmv_hint**(), although it is recommended as the default settings might not be appropriate in every use-case. The optimize phase, **armpl_spmv_optimize**(), might be applied on the first execution of **armpl_spmv_exec_z** if it has not already been explicitly called by the user.

Syntax

C specification:

```
#include "armpl.h"

armpl_status_t armpl_spmv_exec_z(enum sparse_hint_value trans,
                                 armpl_doublecomplex_t alpha,
                                 armpl_spmat_t A,
                                 onst armpl_doublecomplex_t *x,
                                 armpl_doublecomplex_t beta,
                                 armpl_doublecomplex_t *y);
```

Fortran 2003 specification:

```
use armpl_library

subroutine armpl_spmv_exec(trans, alpha, A, x, beta, y, info)
         use armpl_kinds
         implicit none
         integer(kind=armpl_i4), intent(in) :: trans
         complex(kind=armpl_r64), intent(in) :: alpha
         integer(kind=armpl_i8), intent(in) :: A
         complex(kind=armpl_r64), intent(in) :: x(*)
         complex(kind=armpl_r64), intent(in) :: beta
         complex(kind=armpl_r64), intent(inout) :: y(*)
         integer, intent(out) :: info
end subroutine armpl_spmv_exec
```

Returns

In C/C++, this function returns an `armpl_status_t`, which can be tested to check that the function returned correctly. In Fortran, this value is stored in the parameter `info`.

Possible return values are:

- **`ARMPL_STATUS_SUCCESS`** The function returned with no errors.
- **`ARMPL_STATUS_INPUT_PARAMETER_ERROR`** An error was found with the supplied input parameters.
- **`ARMPL_STATUS_EXECUTION_FAILURE`** The function failed for an alternative reason.

Parameters

`trans`

Input parameter

`trans` is `enum sparse_hint_value`

`trans` describes the transpose operation to be performed on the sparse matrix A before calculating the matrix-vector multiplication.

Constraint: `trans` is either `ARMPL_SPARSE_OPERATION_NOTRANS`, `ARMPL_SPARSE_OPERATION_TRANS` or `ARMPL_SPARSE_OPERATION_CONJTRANS` for no transpose, transpose or performing a conjugate transpose, respectively.

`alpha` Input parameter

`alpha` is `armpl_doublecomplex_t`

`alpha` specifies the constant which premultiples the matrix A.

A Input and output parameter

A is armpl_spmat

A is a sparse matrix structure.

Constraint: The sparse matrix A should have been previously created using an appropriate armpl_spmat_create_*_z() function

x Input parameter

x is armpl_doublecomplex_t*

x is the (dense) vector to be multiplied by the matrix A.

Constraint: x is a previously allocated array. This has length determined by the transpose type specified in trans. This length is given in terms of the dimensions of the sparse matrix object created in a armpl_spmat_create_*_z function: namely of length n when trans=ARMPL_SPARSE_OPERATION_NOTRANS and length m otherwise.

beta Input parameter

beta is armpl_doublecomplex_t

beta specifies the scalar to multiply each entry of the vector, y, on input.

Constraint: If beta = 0 then the array y is output only.

y Input and output parameter

y is armpl_doublecomplex_t

y gives the output of the sparse matrix-vector multiplication.

Constraint: y is a previously allocated array. This has length determined by the transpose type specified in trans. This length is given in terms of the dimensions of the sparse matrix object created in a armpl_spmat_create_*_z function: namely of length m when trans=ARMPL_SPARSE_OPERATION_NOTRANS and length n otherwise.

6.3.23 armpl_spmat_destroy

armpl_spmat_destroy frees all memory associated with the supplied Arm Performance Libraries sparse matrix structure.

Syntax

C specification:

```
#include "armpl.h"

armpl_status_t armpl_spmat_destroy(armpl_spmat_t A);
```

Fortran 2003 specification:

```
use armpl_library

      subroutine armpl_spmat_destroy(A, info)
         use armpl_kinds
         integer(kind=armpl_i8), intent(inout) :: A
         integer, intent(out) :: info
end subroutine armpl_spmat_destroy
```

Returns

In C/C++, this function returns an `armpl_status_t`, which can be tested to check that the function returned correctly. In Fortran, this value is stored in the parameter `info`.

Possible return values are:

- **`ARMPL_STATUS_SUCCESS`** The function returned with no errors.
- **`ARMPL_STATUS_INPUT_PARAMETER_ERROR`** An error was found with the supplied input parameters.
- **`ARMPL_STATUS_EXECUTION_FAILURE`** The function failed for an alternative reason.

Parameters

`A` Input and output parameter

`A` is `armpl_spmat`

`A` is a sparse matrix structure.

Constraint: The sparse matrix `A` should have been previously created using an appropriate `armpl_spmat_create_*()` function.

Constraint: The sparse matrix `A` should have been previously created using an appropriate `armpl_spmat_create_*()` function and should not be used again